Vývoj prostředí pro interaktivní aplikaci v MATLABu

Dušan Gavenda

Bakalářská práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Dušan Gavenda**
Osobní číslo:	**A15241**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Vývoj prostředí pro interaktivní aplikaci v MATLABu**
Téma anglicky:	**The Development of an Environment for Interactive Applications in MATLAB**

Zásady pro vypracování:

1. **Zpracujte literární průzkum o programu MATLAB.**
2. **Seznamte se s bakalářskými pracemi, které se zabývají interaktivními aplikacemi v programu MATLAB.**
3. **Sestrojte strukturu programového prostředí pro interaktivní aplikaci.**
4. **Vytvořte vlastní databázi otázek z MATLABu.**
5. **Uveďte popis tvorby interaktivní aplikace a použité funkce.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **KARBAN, Pavel. Výpočty a simulace v programech Matlab a Simulink. Praha: BEN-technická literatura, 2007. ISBN: 978-80-251-1448-3.**
2. **DOŇAR Bohuslav, ZAPLATÍLEK Karel. MATLAB – tvorba uživatelských aplikací. 1. vydání. Praha: BEN – technická literatura, 2004. 216 s. ISBN: 80-7300-133-0.**
3. **DUŠEK, František. MATLAB a Simulink – Úvod do používání. 1.vydání. Pardubice: Univerzita Pardubice, 2001. 146 s. ISBN: 80-7194-273-1**
4. **HECZKO Michal. Výukový a zkušební program pro předmět PPAŘ. Bakalářská práce. Zlín: Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky, 2006.**
5. **KOZÁK, Štefan a Slavomír KAJAN. Matlab – Simulink. 1. vyd. Bratislava: Slovenská technická univerzita, 1999. 125 s. ISBN: 80-227-1213-2.**

Vedoucí bakalářské práce: **Ing. Karel Perůtka, Ph.D.**
Ústav řízení procesů

Datum zadání bakalářské práce: **21. prosince 2018**

Termín odevzdání bakalářské práce: **15. května 2019**

Ve Zlíně dne 21. prosince 2018

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13.5.2019 Dušan Gavenda v.r.

ABSTRAKT

Bakalářská práce se věnuje vývoji prostředí pro interaktivní aplikaci v programu MATLAB. Práce je rozdělena na teoretickou a praktickou část.

V teoretické části práce jsou popsány použité technologie a jiné studentské práce zabývající se interaktivními aplikacemi v MATLABu.

Praktická část práce je zpracována z programátorského i uživatelského hlediska. Jsou v ní popsány všechny funkce aplikace, která je přiložena na disku CD-ROM.

Klíčová slova: MATLAB, Programování, Aplikace

ABSTRACT

This Bachelor thesis deals with the development of environment of interactive application in MATLAB. Thesis is divided into theoretical and practical parts.

In the theoretical part, there is description of used technologies and other student theses dealing with interactive applications in MATLAB.

The practical part is processed from programmatical and user point of view. All functions of application that is attached on CD-ROM are described in this part.

Keywords: MATLAB, Programming, Application

Tímto bych rád poděkoval vedoucímu bakalářské práce, Ing. Karlu Perůtkovi Ph.D., za odborné vedení, rady a návrhy během práce na bakalářské práci.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Cílem této bakalářské práce je vytvoření prostředí pro interaktivní aplikaci v programu MATLAB. Výsledkem je aplikace zpracovaná do podoby 2D hry, kde se uživatel pohybuje po cestě z políček. Některá z těchto políček jsou tzv. bariéry, aktivující se po sešlápnutí. Po sešlápnutí bariéry se zobrazí otázka z prostředí MATLABu, po jejímž zodpovězení bariéra zmizí.

V teoretické části práce je krátký popis programu MATLAB, dále jsou zde popsány jeho funkce, druhy souborů se kterými pracuje, proměnné, datové typy, podmínky, cykly, relační operátory a tvorba grafických interaktivních prostředí.

Teoretická část obsahuje krátké shrnutí funkcí jazyka SQL, který je v programu využit pro práci s databází.

Poslední kapitola teoretické části se zabývá bakalářskými pracemi z minulých let, které se zabývaly podobnými tématy. Jsou zde krátké popisy aplikací a jejich zhodnocení.

Druhá část práce je zaměřena prakticky. V této části jsou popsány specifikace aplikace. Dále je zde rozveden její vývoj, funkce jednotlivých souborů, vývojové diagramy a práce s databází.

Součástí praktické části je také ukázka aplikace z uživatelského pohledu, kde si uživatel před spuštěním samotného programu může udělat jasný obrázek o tom, co ho čeká.

I. TEORETICKÁ ČÁST

1 MATLAB

MATLAB je programové prostředí a skriptovací programovací jazyk vyvíjený firmou MathWorks. [6]

1.1 O programu

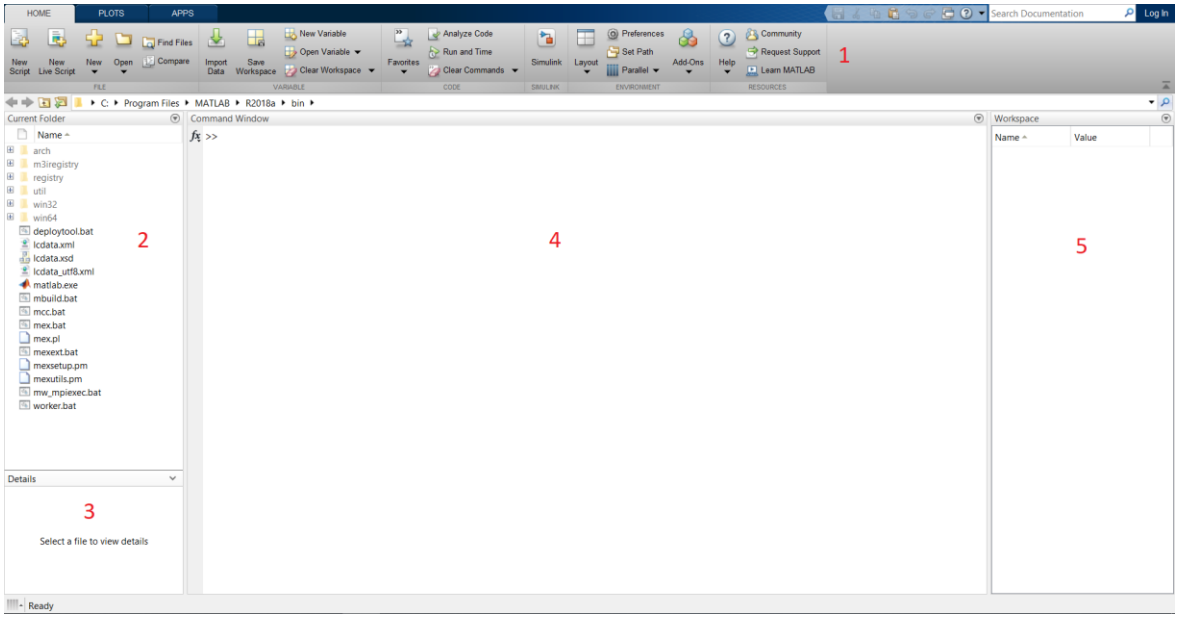
Mezi jeho hlavní výhody patří velké možnosti práce s maticemi, grafy a intuitivní vytváření uživatelských rozhraní. Jeho základním datovým typem je dvourozměrné pole. Tyto vlastnosti z něj tvoří jeden z nejvhodnějších nástrojů pro složité analýzy, měření, modelování, simulace a vizualizaci dat, využít se však dá i pro vytváření algoritmů. [3, 4]

Programovací jazyk MATLABu se v mnoha ohledech dokáže vyrovnat možnostem známějších a používanějších programovacích jazyků jako je Java, C++, C. [6]

1.2 Popis prostředí

Po spuštění programu MATLAB se objeví úvodní obrazovka rozdělená na několik částí:

1. *Menu*
2. *Current Folder*
3. *Details*
4. *Command Window*
5. *Workspace*



Obrázek 1. Úvodní obrazovka MATLABu

1.2.1 **Menu**

Menu se dělí na tři záložky. Každá záložka má jiný účel a obsahuje několik dalších kategorií, pomocí kterých jsou rozděleny jednotlivé funkce.

1.2.1.1 Home

Úvodní záložka obsahující hlavní ovládání a nastavení programu.

- *FILE* – Zde jsou umístěny funkce pro vytváření, otvírání, hledání a porovnávání souborů.
- *VARIABLE* – V této kategorii jsou tlačítka umožňující práci s proměnnými. Obsahuje možnosti jako importování dat, uložení *Workspace* nebo vytváření a otevírání proměnných.
- *CODE* – Umožňuje přístup k dříve použitým příkazům, dále také jejich mazání v *Command Window* a v *Command History*.
- *SIMULINK* – Obsahuje pouze jedno tlačítko spouštějící rozšíření Simulink.
- *ENVIROMENT* – Zde se nachází nastavení programu jako například rozložení oken a nastavení barevného rozhraní.
- *RESOURCES* – Tato kategorie slouží jako nápověda, ať už pomocí samotné dokumentace nebo připojení na komunitní fórum.

1.2.1.2 Plots

Tato záložka se stará o vykreslování grafů.

- *SELECTION* – Okno pro výběr proměnné z *Workspace*.
- *PLOTS* – Tady se nachází velké množství různých grafů, jež lze vykreslit na základě vybrané proměnné.
- *OPTIONS* – V této kategorii si uživatel může vybrat, zda chce graf vykreslit do současného nebo nového okna.

1.2.1.3 Apps

Obsahuje pouze dvě kategorie. Obecně slouží k používání rozšiřujících aplikací programu.

- *FILE* – Slouží k vyhledávání a instalování nových aplikací.
- *APPS* – Zde jsou zobrazeny všechny aktuálně dostupné aplikace.

1.2.2 Current Folder

Jak je patrné z názvu, jedná se o aktuálně vybranou složku. V případě, že spouštíme vytvořený program, skript nebo soubor GUI, tak bude MATLAB právě v téhle složce daný soubor (a jeho případné závislosti) hledat.

1.2.3 Details

V okně *Details* můžeme rychle a jednoduše hledat důležité informace o souboru, který je aktuálně zvolený v *Current Folder*. U souborů MATLABu zde například můžeme spatřit funkce, obsažené v daném souboru.

1.2.4 Command Window

Do tohoto okna se zapisují příkazy a vypisují se jejich výsledky.

1.2.5 Workspace

V okně *Workspace* vidíme všechny aktuálně dostupné proměnné (např. vytvořené v *Command Window).* Můžeme si je rozkliknout a podívat se na jejich detaily (např. obsah matic, vektorů, objektů). V případě potřeby zde můžeme proměnné i měnit nebo mazat. Stejné tak jako jednotlivé proměnné, můžeme smazat nebo vytvořit i celý obsah okna *Workspace*.

1.3 Proměnné

MATLAB automaticky vytváří proměnné při prvním přiřazení hodnoty do neznámého pojmenování, např.:

```
>> promenna = 5

promenna =

           5
```

1.3.1 Lokální proměnné

Jak už název napovídá, jedná se o proměnné, které se vyskytují pouze v aktuálním *Workspace*. Pokud používáme funkce, každá lokální proměnná je dostupná pouze pro funkci, ve které byla vytvořena.

1.3.2 Globální proměnné

Globální proměnná je dostupná pro všechny soubory a funkce v daném *Workspace*. Na rozdíl od lokální proměnné tady již nestačí nechat proměnnou vytvořit automaticky, ale je třeba si ji definovat předem pomocí klíčového slova *global.*

1.4 Datové typy

MATLAB obsahuje velké množství datových typů.

1.4.1 Numerické datové typy

Numerické datové typy se dělí do několika kategorií podle jejich vlastností.

1.4.1.1 Double

Datový typ *double* je základním numerickým datovým typem MATLABu. Jedná se o datový typ pracující s reálnými čísly a plovoucí desetinnou čárkou. V paměti zabírá 8 bytů. [7]

1.4.1.2 Single

Single má podobné vlastnosti jako *double,* na rozdíl od typu *double* – využívá pouze poloviční přesnost, díky které je i jeho velikost v paměti poloviční, tedy 4 byty. [7]

1.4.1.3 Uint

MATLAB nabízí 4 datové typy *uint – uint8*, *uint16*, *uint32*, *uint64*. *Uint* je zkratka pro unsigned integer a jedná se o nezáporná celá čísla. Jejich velikost v paměti je ve všech variantách menší než u *double* a *single.* Čísla za zkratkou značí rozsah čísel, která daný datový typ může obsahovat, např. *uint8* má rozsah od 0 do 255 a zabírá 8 bitů. [7]

1.4.1.4 Int

Stejně jako u *uint*, tak i tady jsou k dispozici 4 možnosti - *int8*, *int16*, *int32*, *int64.*Na rozdíl od *uint* zde můžeme využít záporných čísel, rozsah je dán počtem bitů a je rovnoměrně rozložen kolem počátku. Např. datový typ *Int8* má rozsah od -128 do 127 a také zabírá 8 bitů. [7]

1.4.2 **Textové datové typy**

V MATLABu se objevují dvě možnosti zápisu textu.

1.4.2.1 Char

Datový typ *char* slouží pro uložení jednoho znaku a do proměnné se ukládá znak uvozený pomocí apostrofů.

1.4.2.2 String

Proměnná datového typu *String* v sobě na rozdíl od datového typu *char* může nést neomezené množství znaků a definuje se pomocí uvozovek.

1.4.3 **Struct**

Struktura je speciální datový typ, použitelný v případě, kdy potřebujeme mít pohromadě několik proměnných různých datových typů (např. *double* a *string*). [7]

Struktura sama o sobě pouze shromažďuje proměnné pomocí polí (*fields*), ke kterým se přistupuje pomocí tečky. [1,7]

Můžeme ji definovat například pomocí příkazu:

```
>> struktura = struct(jmeno, "name", cislo, 2)
```

Tento příkaz vytvoří proměnnou *struktura*, která bude mít 2 pole, jeden z nich je datového typu *string* a druhý je typu *double*. K hodnotám jejich obsahu se můžeme dostat díky následujícím příkazům:

```
>> struktura.jmeno
```

```
>> struktura.cislo
```

Stejným postupem bychom mohli vytvořit jakékoliv množství polí. [1]

1.4.4 **Table**

Tabulka se podobně jako struktura použije v případě, kdy chceme mít hodnoty různých datových typů v jedné sadě pohromadě. Má několik integrovaných parametrů jako jsou názvy proměnných a řádků, popisek a také možnost si nastavit vlastní parametry. [7]

Je to základní datový typ pro práci s databází.

Pro vytvoření tabulky se dvěma proměnnými můžeme použít příkaz:

```
tabulka = table(cislo, text, 'VariableNames', ...
{'Čísla', 'Texty'});
```

1.5 Pole a jeho variace

Všechny proměnné, které jsou v MATLABu vytvořeny se automaticky ukládají jako pole. Při ukládání jednoho prvku se vytvoří pole o rozměrech 1x1. Právě práce s poli je jedna z největších výhod MATLABu, protože na rozdíl od ostatních programovacích jazyků je MATLAB stavěný primárně na práci na jednorozměrných a dvourozměrných polích. [7]

1.5.1 **Definice**

Pole se definují pomocí hranatých závorek, kde prvky na jednom řádku oddělují pomocí čárky nebo mezery a sloupce se oddělují pomocí středníku.

1.5.1.1 Vektor

Jako *vektor* se označuje jednorozměrné pole, tedy pole s jedním řádkem a několika sloupci nebo jedním sloupcem a několika řádky. [7]

```
>> vektor = [1 2 3 4]
vektor =
     1    2    3    4
```

Na předchozím příkladu lze vidět řádkový vektor o velikosti 1x4 s datovým typem double.

Sloupcový vektor můžeme vytvořit, pokud mezi jednotlivá čísla vložíme středníky.

```
>> vektor = [1; 2; 3; 4]
vektor =
     1
     2
     3
     4
```

Jak lze vidět, tak tento příkaz nám vytvořil sloupcový vektor o velikosti 4x1.

1.5.1.2 Matice

Typ *matice* je dvourozměrné pole, tedy pole s počtem řádků i sloupců větším než 1. Vytvoření matice o velikosti 3x3:

```
>> matice = [1 2 3; 4 5 6; 7 8 9]

matice =

     1     2     3

     4     5     6

     7     8     9
```

1.5.2 **Automatické generování pole**

MATLAB nabízí velkou škálu možností jak si ulehčit práci při vytváření polí.

K jednoduchému vytvoření aritmetické posloupnosti od jedné do 7 nám vystačí dvojtečka.

```
>> vektor = 1:7

vektor =

     1     2     3     4     5     6     7
```

Další dvojtečkou lze vloženou doprostřed předchozího příkazu lze specifikovat vzdálenost čísel, která se budou generovat. K vygenerování lichých čísel stačí nastavit počáteční index na liché číslo a krok generování na dva:

```
>>vektor = 1:2:7

vektor =

     1     3     5     7
```

V MATLABu existují i funkce pro automatické generování polí. Zde jsou některé z nich:

zeros(n, m) – Vygeneruje matici o *n* řádcích a *m* sloupcích naplněnou nulami. [7]

rand(n, m) – Vygeneruje matici naplněnou náhodnými čísly od 0 po 1. [7]

randi([imin, imax], n, m) – Vygeneruje matici naplněnou náhodnými celými čísly od *imin* po *imax*. [7]

randperm(n, k) – Náhodná permutace. Vygeneruje vektor *k* prvků v hodnotách od 1 po *n*, kde se každá hodnota vyskytuje pouze jednou. [7]

1.5.3 **Přístup k prvkům**

MATLAB nabízí velmi intuitivní přístup k prvkům *vektoru* nebo *matice*. Pokud máme vytvořený vektor, pak stačí pouze definovat index prvku do kulaté závorky.

```
>>vektor(2)

     ans =

          2
```

K prvkům matice se přistupuje stejným způsobem, jen je třeba do závorky definovat index druhého rozměru a oddělit ho od prvního indexu čárkou. Prvním indexem se tedy definuje řádek a druhým sloupec prvku, který se hledá.

```
>>matice(3, 2)

     ans =

          8
```

Matice je v MATLABu pouze vícerozměrným *vektorem*, to znamená, že hodnoty jsou uloženy řádek po řádku, a i když právě pracujeme s *maticí*, lze prvky vypisovat stejně jako bychom pracovali s *vektorem*.

Číslo na třetím řádku a ve druhém sloupci je uloženo jako osmé číslo *vektoru*. Vypsat ho tedy můžeme pomocí příkazu:

```
>>matice(8)
```

Pro zobrazení než jednoho prvku *matice* můžeme využít dvojtečky, například pro vyhledání všech prvků daného sloupce stačí nahradit dvojtečkou index řádku:

```
>>matice(2, :)

     ans =

          4     5     6
```

Pomocí dvojtečky můžeme velmi specificky definovat prvky, které v *matici* hledáme. Pokud hledáme například prvky vyskytující se od prvního, po druhý řádek, můžeme jedničku a dvojku na indexu řádku oddělit dvojtečkou, stejně tak u sloupců.

```
>>matice(1:2, 2:3)

     ans =

          2     3

          5     6
```

Předcházející případ vrací všechny prvky vyskytující se od prvního po druhý řádek, a od druhého po třetí sloupec.

1.5.4 **Operace**

V operacích s poli má MATLAB oproti klasickým programovacím jazykům spoustu velkých výhod. Zatímco v některých programovacích jazycích by se pro tyto činnosti musely používat cykly, MATLAB poskytuje spoustu integrovaných funkcí a velmi zjednodušuje práci s poli.

Pro sečtení prvků *matice* nebo *vektoru* stačí využít stejného příkazu, jako bychom pracovali s jediným prvkem:

```
>> matice = matice1 + matice2
```

MATLAB v předchozím případě vezme každý prvek první matice a sečte ho s korespondujícím prvkem druhé matice. Stejně tak můžeme ke každému prvku *matice* například přičíst číslo:

```
>> matice = matice + 2
```

Pro nalezení transponované *matice* lze jednoduše použít apostrof.

```
>> matice = matice'
```

Pro násobení matic jsou v jiných programovacích jazycích potřeba algoritmy, které postupně projdou všechny řádky a sloupce matice a všechny čísla korektně vynásobí. V MATLABu nám k tomuhle stačí jednoduchý operátor. [7]

```
>> matice = matice1 * matice2
```

Pokud je z nějakého důvodu třeba vynásobit každý prvek s první matice s korespondujícím prvkem druhé matice stejně jako u sčítání, MATLAB opět nabízí jednoduché řešení.

```
>> matice = matice1 .* matice2
```

1.6 Druhy souborů

MATLAB obsahuje několik druhů souborů, každý z nich má jinou příponu.

1.6.1 **M soubory**

Pro jednoduché jednořádkové výpočty lze použít *Command Window*. Pro sadu příkazů a, případně celé algoritmy, je vhodné kód ukládat. K tomu slouží právě M soubory, které jsou uloženy na disku a k jejich použití stačí do *Command Window* napsat jejich název (pokud se soubor nachází ve složce, ve které MATLAB vyhledává).

1.6.2 **FIG soubory**

FIG soubory uživatel využije především při vytváření grafického uživatelského rozhraní GUI. Tyto soubory v sobě mají uloženy informace o obsahu, parametrech a nastavení GUI.

1.7 Možnosti psaní kódu

V MATLABu lze psát kód dvěma hlavními způsoby.

1.7.1 **Skripty**

Skripty jsou posloupnosti příkazů uloženy v M souborech. Lze je spustit napsáním jejich názvu do *Command Window*. Jejich hlavní výhodou je jednoduchost použití. Nehodí se pro psaní složitých algoritmů, protože jim chybí určité schopnosti, které jsou často potřeba, například je nutné opětovné načítání do paměti při každém volání souboru. [2, 7]

1.7.2 **Funkce**

Funkce mají na rozdíl od skriptů mnohem univerzální využití díky jejich schopnosti přijímat argumenty a vracet hodnoty. Můžeme je deklarovat pomocí:

```
function [navratovaHodnota] = nazevFunkce(argument)
```

V MATLABu se objevují dva druhy funkcí. [7]

1.7.2.1 Globální funkce

Globální *funkce* mají jednu hlavní výhodu – lze je volat odkudkoliv (nachází-li se v adresáři, ve kterém MATLAB hledá). Jako globální se označuje *funkce*, která se jmenuje stejně jako soubor, ve kterém je uložena.

1.7.2.2 Lokální funkce

Jak už naznačuje název, jedná se o *funkci* dostupnou pouze uvnitř daného souboru. Lokální *funkce* lze dosáhnout přesně naopak než globální funkce – nepojmenovat jí podle názvu souboru, ve kterém je uložena. Funkce je následně dostupná pouze z kódu, který je uvnitř stejného souboru.

1.8 Operátory

Stejně jako v jiných programovacích jazycích, i zde se setkáme s množstvím logických a relačních operátorů.

Tyto operátory se obvykle používají pro větvení kódu pomocí podmínek.

1.8.1 **Relační operátory**

Tyto operátory slouží k porovnání hodnot. Návratová hodnota porovnání dvou proměnných je vždy *true* (pravda) nebo *false* (nepravda). Mezi tyto operátory patří:

< - menší než

> - větší než

<= - menší nebo rovno než

>= - větší nebo rovno než

== - rovná se

~= - nerovná se

1.8.2 Logické operátory

Logické operátory jsou v MATLABu o něco pokročilejší než ve většině jiných jazyků. Stejně jako jinde lze i zde použít logický součet v podmínce (||), logický součin v podmínce (&&), a také negaci (~).

Navíc se zde ještě objevuje logický součet pro práci s polem (|) a logický součin pro práci s polem (&).

1.9 Větvení kódu

Stejně jako ve většině programovacích jazyků, můžeme i v MATLABu naleznout dva hlavní způsoby větvení kódu.

1.9.1 Podmínky if, else, elseif

Se základní znalostí angličtiny lze jednoduše zjistit, co mají podmínky za úkol. *If* tj. *jestliže* platí podmínka tak se dělá tato větev a *jinak* tj. *else* se provádí tato část. Například

```
if cislo == 5
     disp('Číslo je 5');
elseif číslo < 5
     disp('Číslo je menší než 5');
else
     disp('Číslo je větší než 5');
end
```

Předchozí příklad vypisuje text v závislosti na obsahu proměnné `cislo`.

MATLAB poskytuje možnost psát podmínky donekonečna. Z programátorského hlediska je však důležité udržovat kód co možná nejpřehlednější a vyhnout se tak velkému množství podmínek.

1.9.2 Logický přepínač switch

Switch je skvělým nástrojem v případě, že se chystáme vykonat spoustu různých příkazů na základě hodnoty jedné proměnné.

Switch vykonává příkazy na základě proměnné, která je definovaná hned za deklarací přepínače. Hodnotu této proměnné použije pro porovnání s hodnotami uvedenými v *case*, pokud existuje shoda, tak se vykonají příkazy, které jsou uvedeny pod příslušným *case*, jinak se provedou příkazy definovány v *otherwise*. V případě, že *otherwise* nepoužijeme a *switch* nenajde shodu u žádného case, neprovede se žádný příkaz, čímž se liší od *if else*.

Pro *switch*, který bude vykonávat tři různé příkazy na základě toho, zda se hodnota proměnné *cislo* rovná dané hodnotě můžeme psát následující kód:

```
switch cislo
     case 1
          disp('Číslo je 1');
     case 2
          disp('Číslo je 2');
     otherwise
          disp('Číslo není 1 ani 2');
end
```

1.10 Cykly

Využívat můžeme dva druhy cyklů, každý slouží k jiným účelům.

1.10.1 For

Cyklus *for* se využívá v případech, kdy se předem ví množství potřebných průchodů. Klasickým příkladem by bylo naplnění polí určité velikosti.

Cyklus jde ukončit předčasně pomocí příkazu *break* (může se například vytvořit podmínka, která spustí příkaz *break* za určitých okolností).

```
for i = 1:10
     disp(['Číslo průchodu je ' num2str(i)]);
end
```

1.10.2 **While**

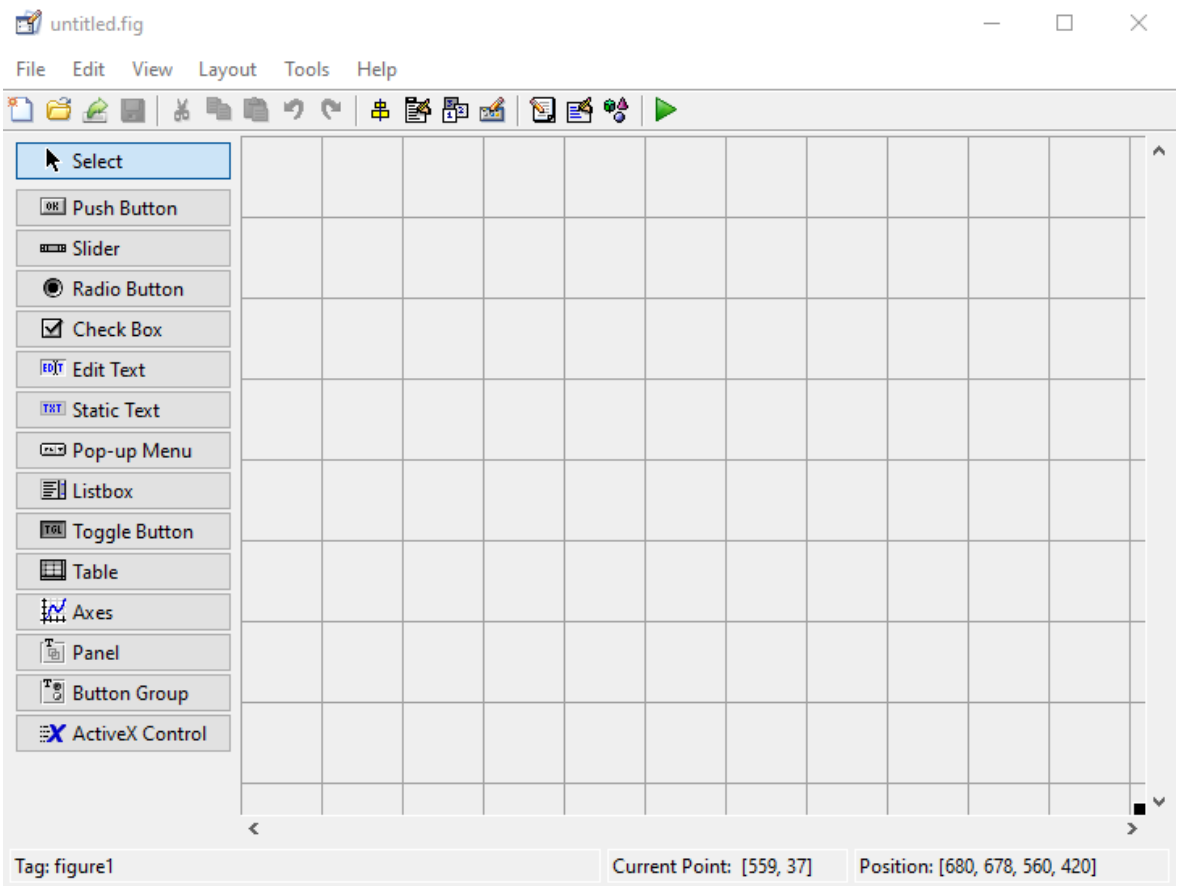
Cyklus *while* se na rozdíl od *for* použije v případě, kdy je nutné, aby cyklus probíhal nepřetržitě do doby, kdy se stane nějaká událost (např. proměnná dosáhne hodnoty). *While* bude probíhat do doby, dokud definovaný výraz vrací *true*. Stejně jako u *for*, i zde můžeme vynutit předčasné ukončení cyklu pomocí příkazu *break*.

```
while cislo < 10
     disp('Číslo průchodu je menší než 10');
end
```

1.11 Grafické uživatelské rozhraní GUI

MATLAB nabízí velmi intuitivní způsob vytváření grafických uživatelských rozhraní, zkráceně GUI. Všechny informace o GUI jsou uloženy ve FIG souborech.

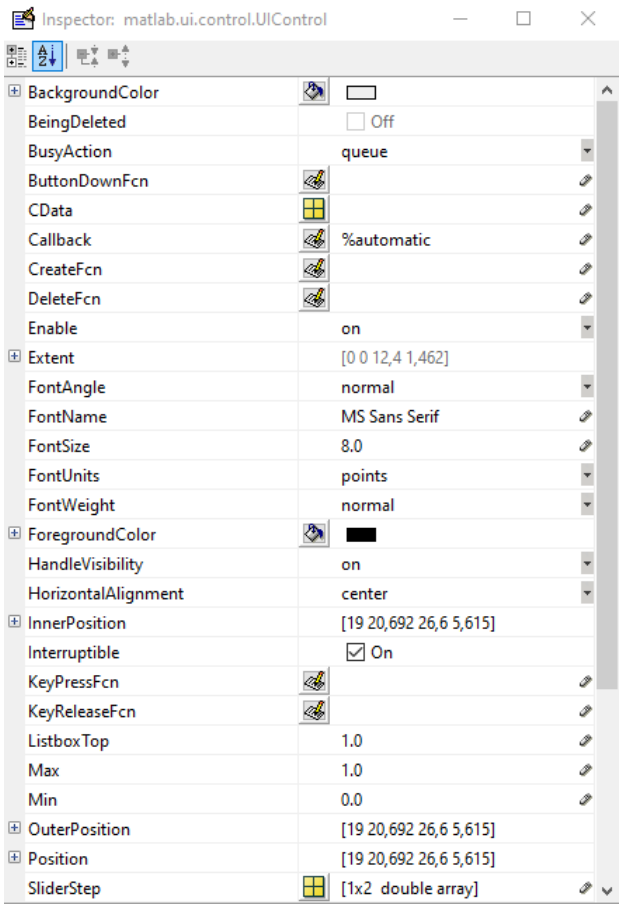
Příkazem *guide* lze vyvolat nabídku možností vytvoření GUI. V nabídce si lze vybrat z přednastavených GUI nebo z prázdné varianty, případně lze také otevřít již vytvořené rozhraní. Po výběru se otevře editor, ve kterém si uživatel může vybrat ze čtrnácti použitelných prvků.



Obrázek2 Příklad prázdného editoru GUI

Uživatel může v editoru jednoduše vytvářet nové prvky, upravovat stávající prvky a jejich parametry, ukládat, načítat nebo spouštět grafické uživatelské rozhraní.

Důležitou vlastností editoru je možnost upravovat parametry jednotlivých prvků nebo klidně celého vytvořeného rozhraní. To lze provést pravým kliknutím na daný prvek a zvolení záložky *Property Inspector*.



Obrázek 3 Náhled *Property Inspector*

1.11.1 **Callback funkce**

Pokud si uživatel neupraví nastavení, pak se po uložení kromě FIG souboru vygeneruje ještě M soubor, který při základním nastavení obsahuje tzv. *callback* (tato se aktivuje v momentě, kdy je kliknutou na daný prvek) funkce všech prvků, na které lze v GUI kliknout včetně funkce pro samotný GUI.

Uživatel může pro každý prvek manuálně vytvořit dodatečné *callback* funkce. Mezi tyto funkce patří:

ButtonDownFcn – Spustí se, pokud uživatel klikne dovnitř prvku nebo 5 pixelů vedle.

CreateFcn – Inicializuje prvek při jeho vytvoření a následně se spustí (před zobrazením prvku).

DeleteFcn – Spustí tzv. *uklízecí* operace před smazáním prvku.

KeyPressFcn – Vykoná se, pokud uživatel zmáčkne tlačítko na klávesnici a je vybraný prvek daného *callbacku*.

ResizeFcn – Vykoná se, když uživatel změní velikost panelu nebo skupiny tlačítek.

CloseRequestFcn – Spustí se při vypnutí GUI.

Všechny parametry GUI jsou ve vygenerovaném M souboru dostupné přes strukturu *handles.*[12]

1.11.2 **Kódové nastavení parametrů GUI**

Parametry, které uživatel může nastavit v *Property Inspectoru*., a které jsou uloženy ve struktuře *handles* lze následně získat pomocí funkce *get* a nastavit pomocí funkce *set*.

V GUI, které obsahuje tlačítko se jménem *tlacitko*, můžeme následujícím příkazem zjistit, zda je viditelné:

```
get(handles.tlacitko, 'Visible')
```

```
     ans =

          'on'
```

Z předchozího příkazu lze vidět, že tlačítko je viditelné, následně ho můžeme s využitím příkazu *set* nastavit na neviditelné.

```
set(handles.tlacitko, 'Visible', 'off')
```

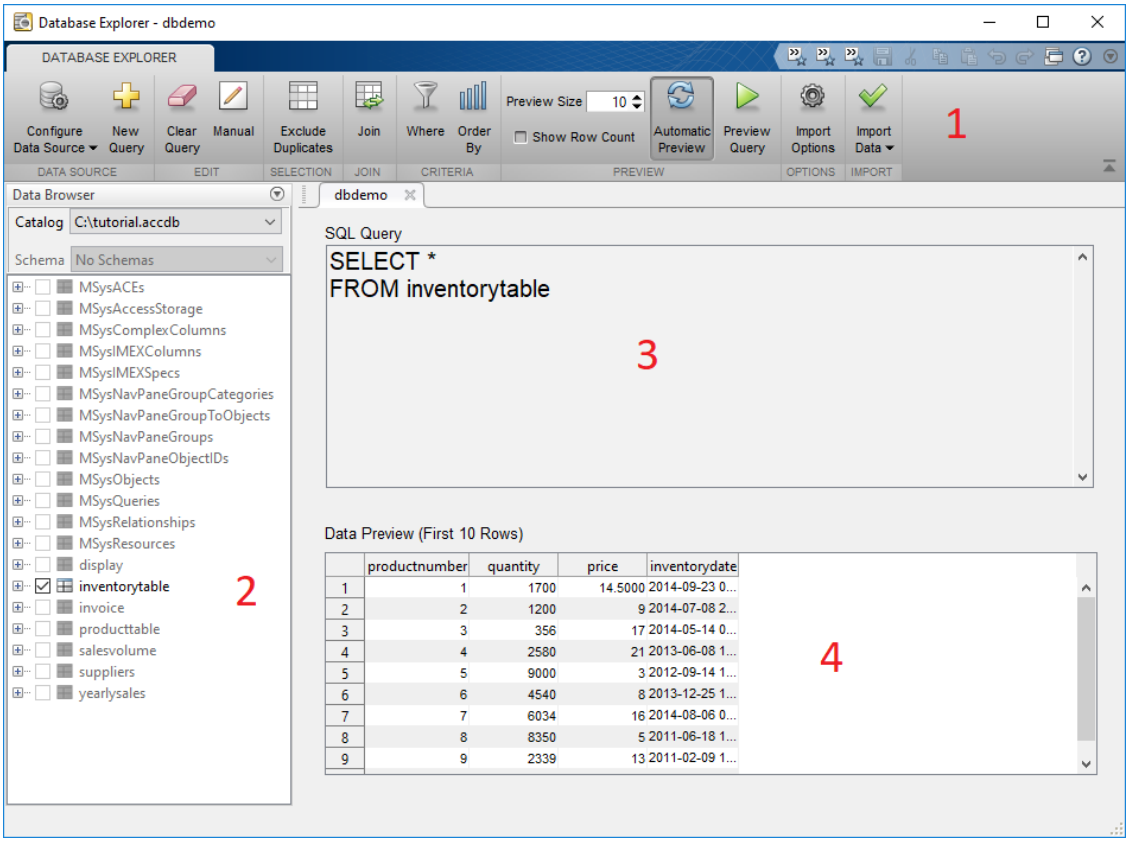
Stejným způsobem můžeme přistupovat ke všem parametrům struktury *handles*.

1.12 Database Explorer

MATLAB poskytuje integrovanou aplikaci pro nastavení, prohlížení a importování dat do programu. Jedná se o velmi jednoduchou a uživatelsky přístupnou možnost jak propojit uživatelskou aplikaci s databází. [7]

Database Explorer lze spustit pomocí záložky *Apps => Show more => Database Connectivity and Reporting => Database Explorer*. Další možností je napsání příkazu *databaseExplorer* do *Command Window*. [7]

Po připojení k databázi může uživatel jednoduchým způsobem vybrat tabulku, jejíž data chce zobrazit.



Obrázek 4 Ukázka *Database Exploreru* s vybranými daty [11]

1. *Menu* – Obsahuje nastavení *Database Exploreru*.
2. *Data Browser* – Zde jsou zobrazeny všechny tabulky v připojené databázi.
3. *SQL Query* – Tady se zobrazuje vygenerovaný *SQL* kód.
4. *Data Preview* – Ukázka dat, získaných z databáze, na základě skriptu z *SQL Query*.

Data, která jsou viditelná v *Data Preview* je možné importovat do *Workspace* pomocí tlačítka *Import Data* v Menu. Tato data jsou následně uložena v proměnné jménem *data*, datového typu *table*.

2 SQL

SQL je zkratkou pro *Structured Query Language* (v překladu strukturovaný dotazový jazyk). Používá se pro komunikaci s databází. Jedná se o standard jazyku pro vztahové databázové systémy. [10]

SQL výrazy jsou využívány k provádění úkolů, jako je aktualizace nebo získávání dat z databáze. Mezi používané databázové systémy využívající SQL patří *Microsoft SQL server* a *Oracle*. [10]

Účelem SQL je poskytnutí rozhraní pro databázové systémy, které na tomto jazyku mohou stavět a rozšiřovat jeho funkcionalitu pro svoje účely. Na rozdíl od většiny programovacích jazyků, SQL nezpracovává data jednotlivě, ale po skupinách. [9]

Přestože většina databázových systémů využívá SQL, většina z nich zároveň využívá jejich vlastní doplňky, které jsou využívány pouze na jejich systémech. Pro drtivou většinu potřebných funkcí databáze si však běžný uživatel vystačí se základními příkazy, které poskytuje SQL jazyk. [10]

2.1 Příkazy

SQL obsahuje několik důležitých příkazů, které se využívají stále dokola.

2.1.1 **Create, Drop**

Před vracením dat z databáze je třeba vytvořit tabulku, ve které budou data uložena. K tomu slouží příkaz *Create*. [10]

```
CREATE TABLE Uzivatel(
Jmeno [nvarchar](50) NULL,
Prijmeni [nvarchar](50) NULL)
```

Předchozí příkaz vytvořil tabulku *Uzivatel* se dvěma sloupci, oba sloupce mají datový typ *nvarchar*, což je textový datový typ a záznamy v nich mohou být *null* (prázdné).

Vytvořenou tabulku lze jednoduše smazat pomocí příkazu *Drop*. [10]

```
DROP TABLE Uzivatel
```

2.1.2 **Insert**

Pro vkládání dat do vytvořené tabulky se využívá příkazu *Insert*. Pro přidání dvou řádků do tabulky uživatel lze využít příkazu: [10]

```
INSERT INTO TABLE Uzivatel(Jmeno, Prijmeni)
     VALUES('Jmeno1', 'Prijmeni1'),('Jmeno2', 'Prijmeni2')
```

2.1.3 **Update**

Záznamy v databázi je třeba často upravovat. K tomu slouží příkaz *Update*. Tento příkaz potřebuje informace o tom, který záznam upravit a o tom, který sloupec tohoto záznamu upravit. [10]

```
UPDATE Uzivatel
     SET Jmeno = 'NoveJmeno'
     WHERE Jmeno = 'Jmeno1'
```

Lze si všimnout, že v tomto příkazu se využilo klíčového slova *Where*, pomocí kterého můžeme specifikovat podmínky, podle kterých má databáze vybrat záznamy k úpravě.

2.1.4 **Delete**

Pro mazání záznamů v databázi lze využít *Delete*. [10]

```
DELETE FROM Uzivatel
     WHERE Jmeno = 'NoveJmeno'
```

Stejně jako u Update, i tady je třeba specifikovat podmínky, podle kterých se specifikuje záznam ke smazání, SQL by jinak provedlo příkaz pro celou tabulku.

2.1.5 **Select**

Select neboli výběr je nejpoužívanějším příkazem SQL. Tento příkaz vrací data, jež si uživatel definuje. Ke vrácení všech dat v tabulce *Uzivatel* lze použít příkaz: [10]

```
SELECT Jmeno, Prijmeni
FROM Uzivatel
```

3 JINÉ BAKALÁŘSKÉ PRÁCE ZABÝVAJÍCÍ SE INTERAKTIVNÍMI APLIKACEMI V MATLABU

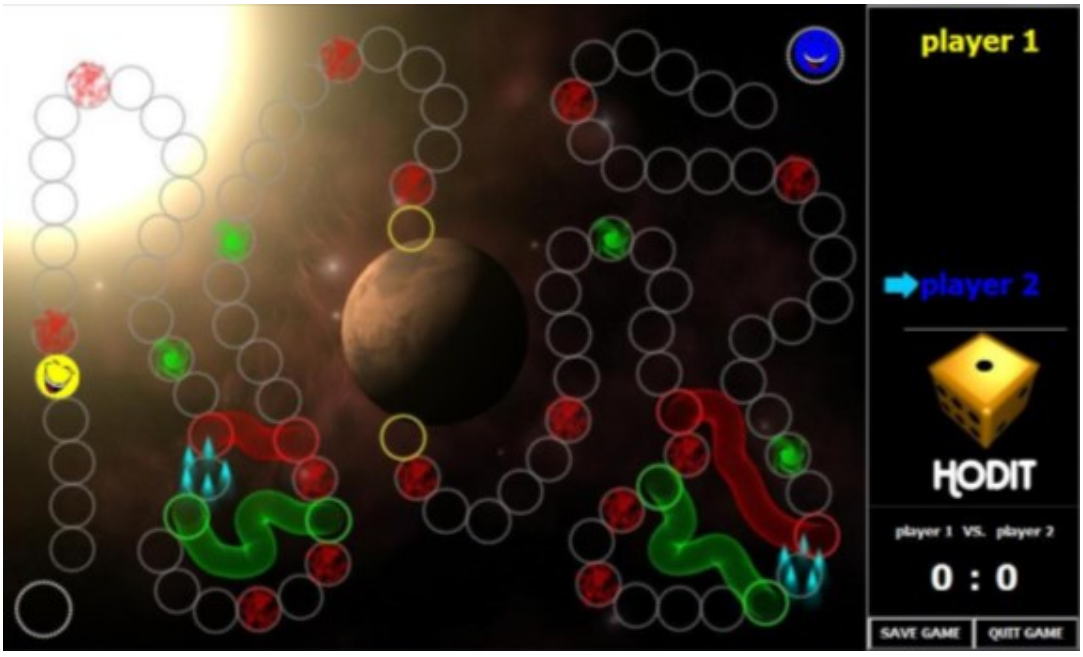
Jedním z důležitých bodů mé práce je seznámení se s jinými bakalářskými pracemi na podobné téma.

3.1 Fields Frenzy

Jedná se o interaktivní aplikaci zpracovanou v podobě deskové hry pro dva hráče. Jejím autorem je Petr Hruboš, který ji zpracoval jako bakalářskou práci v roce 2009.

Desková hra je zpracována do podoby jedné cesty, tvořené z políček, po které se hráči pohybují pomocí hodu kostkou. Na cestě je několik teleportů, při jejichž sešlápnutí se hráč posouvá dopředu nebo dozadu. Hra končí v momentě, kdy jeden z hráčů dosáhne finálního políčka. Rozehrané hry je možné ukládat a načítat. Hra je velmi pěkně audiovizuálně zpracována.

Kódová část aplikace je tvořena z 65 M souborů a grafickou stránku doplňuje spousta dalších grafických a audio souborů.



Obrázek 5 Ukázka ze hry Fields Frenzy

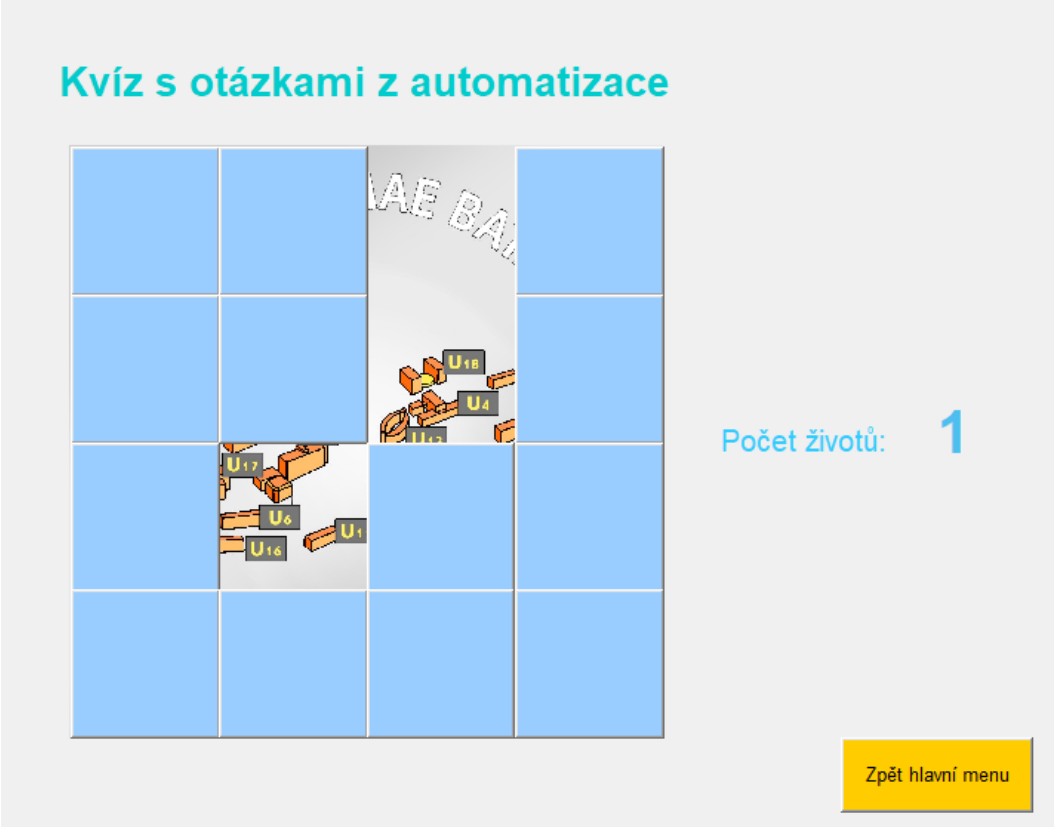
3.2 Kvíz s otázkami z automatizace

Tuto aplikaci vytvořila v roce 2018 Petra Ducháčková. Jedná se o výukovou aplikaci, jíž úkolem je podpořit výuku automatizace v podobě otázkového kvízu.

Kvíz je zpracován v podobě dvanácti interaktivních polí. Po kliknutí se zobrazí otázka z automatizace se čtyřmi možnými odpověďmi, z nichž pouze jedna je správná. Po zodpovězení otázky se obrazovka vrátí zpět na kvíz, kde zmizelo pole, na které se v minulém kole kliknulo. Postupným zodpovídáním otázek se odkrývá skrytý obrázek. S každou špatnou odpovědí se odečítá jeden život (na začátku jsou tři) a hra končí v momentě, kdy hráč přijde o všechny životy, nebo kdy odkryje celý obrázek.

V menu aplikace je možnost si zobrazit vzorové příklady, kde jsou vyřešeny některé příklady z automatizace, včetně jejich názorného vyřešení pomocí MATLABu.

Aplikace je tvořena několika hlavními M a FIG soubory a spoustou menších M souborů, například pro každou otázku zvlášť. Otázky jsou uloženy v textových souborech. Z programátorského hlediska by se tedy aplikace rozhodně dala vyřešit mnohem elegantněji.



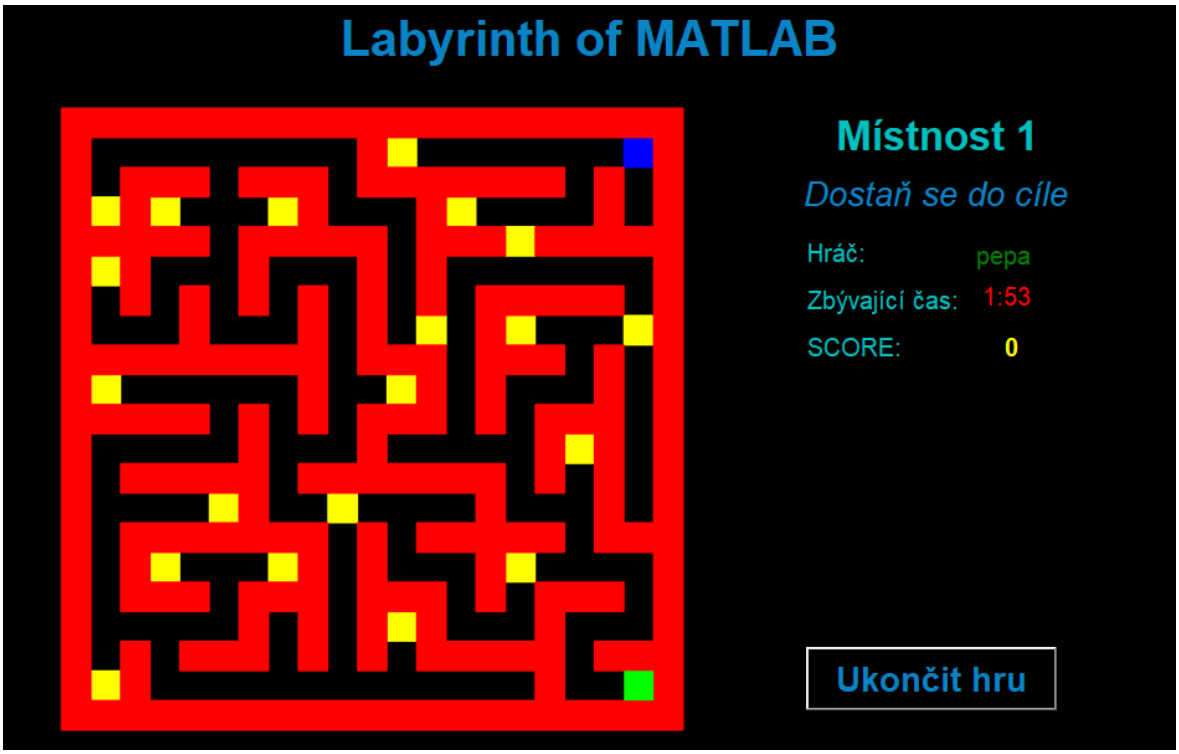
Obrázek 6 Ukázka aplikace Kvíz s otázkami s automatizace

3.3 Labyrinth of Matlab

Aplikaci *Labyrinth of Matlab* byla vytvořena v roce 2016, její autorkou je Lucie Šarmanová.

Tato aplikace je zpracována, jak už název napovídá, do podoby hry, která je tvořena z labyrintu, ve kterém se hráč pohybuje pomocí šipek na klávesnici. V labyrintu jsou barevně zvýrazněna políčka, která vyvolají zobrazení otázky týkající se MATLABu, v případě, kdy na ně hráč stoupne. Po správném zodpovězení otázky políčko zmizí a hráč může postupovat v labyrintu dál. Pokud zodpoví špatně, tak se z políčka stane neprostupná zeď. Hra obsahuje 10 úrovní, a každá úroveň je časově omezena. Hry je možno ukládat a načítat.

Aplikace je programově zpracována do dvou hlavních M a FIG souborů. Dalších 8 menších M souborů je využito pro opakující se funkce jako je ovládání nebo vykreslování labyrintu. Otázky, rozehrané hry a výsledky jsou ukládány do CSV souborů, které zde plní roli databáze.



Obrázek 7 Ukázka hry Labyrinth of Matlab

II. PRAKTICKÁ ČÁST

4 SPECIFIKACE HRY

4.1 Popis hry

MATLAB Quiz je hra určena pro jednoho hráče s dobrými znalosti MATLABu. Je rozdělena na čtyři úrovně se zvyšující se obtížností. Každá úroveň obsahuje cestu tvořenou z políček, po kterých se hráč pohybuje pomocí hodu kostkou.

Některá políčka jsou modře zvýrazněna, tyto políčka označují přítomnost bariéry, na kterou hráč musí stoupnout. Po stoupnutí na bariéru se zobrazí okno s otázkou a čtyřmi odpověďmi z oblasti MATLABu, z nichž pouze jedna je správná. Každá otázka je časově omezena. S každou úrovní se snižuje množství času na otázku a zvyšuje množství bariér.

MATLAB Quiz může hrát kdokoliv, pro úspěšné dokončení hry je však potřeba mít dobré znalosti MATLABu.

Autor	Dušan Gavenda
Určeno pro	Lidé s dobrými znalostmi MATLABu
Počet hráčů	1
Trvání hry	Maximálně 15 minut

Tabulka 1 Parametry hry

4.2 Pravidla hry

Cílem hry je projít čtyřmi úrovněmi. Každá úroveň obsahuje cestu tvořenou z políček, která mají troje různé barevné rozlišení. Na celou hru má hráč k dispozici dohromady 3 životy.

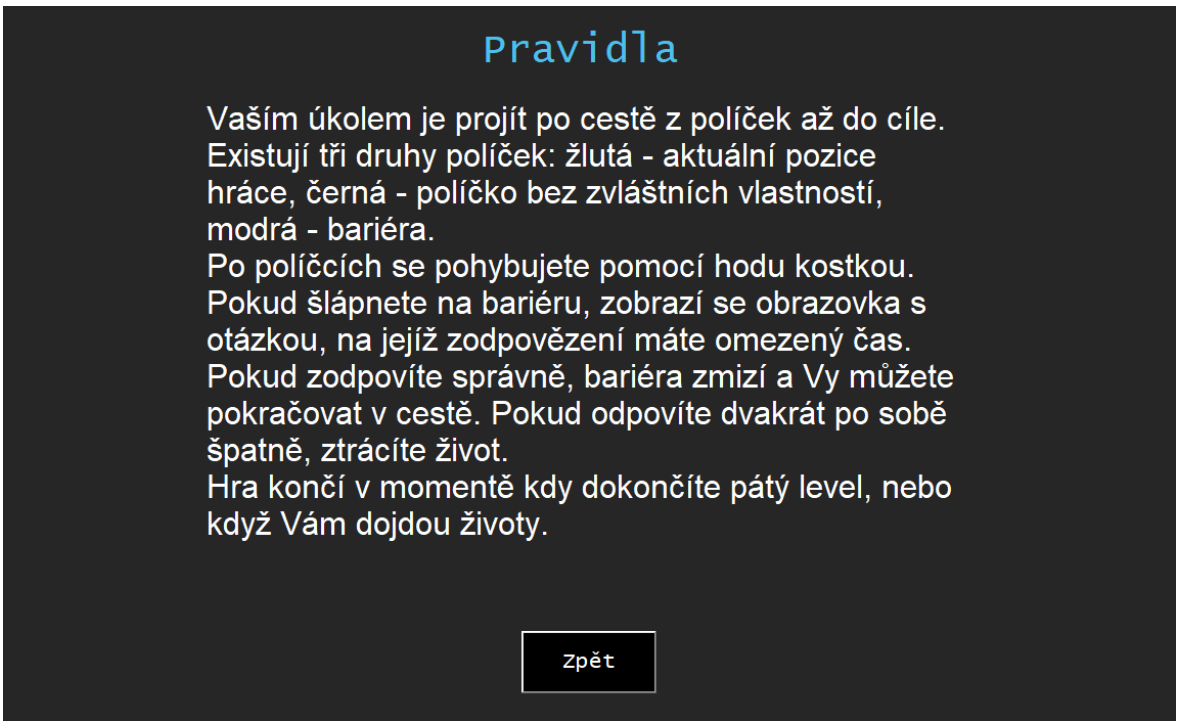
Žlutá značí aktuální pozici hráče. Černá značí políčko bez speciálních vlastností, při stoupnutí na něj se nic nestane. Modrá značí políčko s bariérou, na které musí hráč bezpodmínečně stoupnout.

Po cestě se hráč pohybuje pomocí hodu kostkou. Pokud hodí více než je potřeba k tomu, aby stoupnul na nejbližší bariéru, je vyzván k opakování svého hodu. Pokud hodí méně než je nutné ke stoupnutí na bariéru, posune se o příslušný počet políček. Pokud mu padne číslo, potřebné k tomu, aby se postavil na bariéru, zobrazí se nové okno.

V tomto okně je zobrazen obrázek s otázkou a čtyřmi odpověďmi a časovým limitem. Hráč musí v časovém limitu kliknout na správnou odpověď. Pokud zodpoví špatně, nebo nezodpoví vůbec, dostane ještě jednu otázku. Pokud opět zodpoví špatně, ztrácí život a vrací se zpět na pole. Pokud zodpoví správně, vrátí se na pole bez ztráty života.

Úroveň je dokončena po stoupnutí na poslední políčko. Hra končí v momentě, kdy hráči dojdou životy, nebo když dokončí poslední úroveň.

Za každou správně zodpovězenou otázku se hráči přičítají body. K těmto bodům se na konci hry přičtou dodatečné body za úroveň, do které se hráč dostal, a uloží se do databáze. Pokud je výsledek mezi deseti nejlepšími, tak se následně zobrazí v tabulce Nejlepší hráči.



Obrázek 8 Pravidla hry

5 REALIZACE HRY

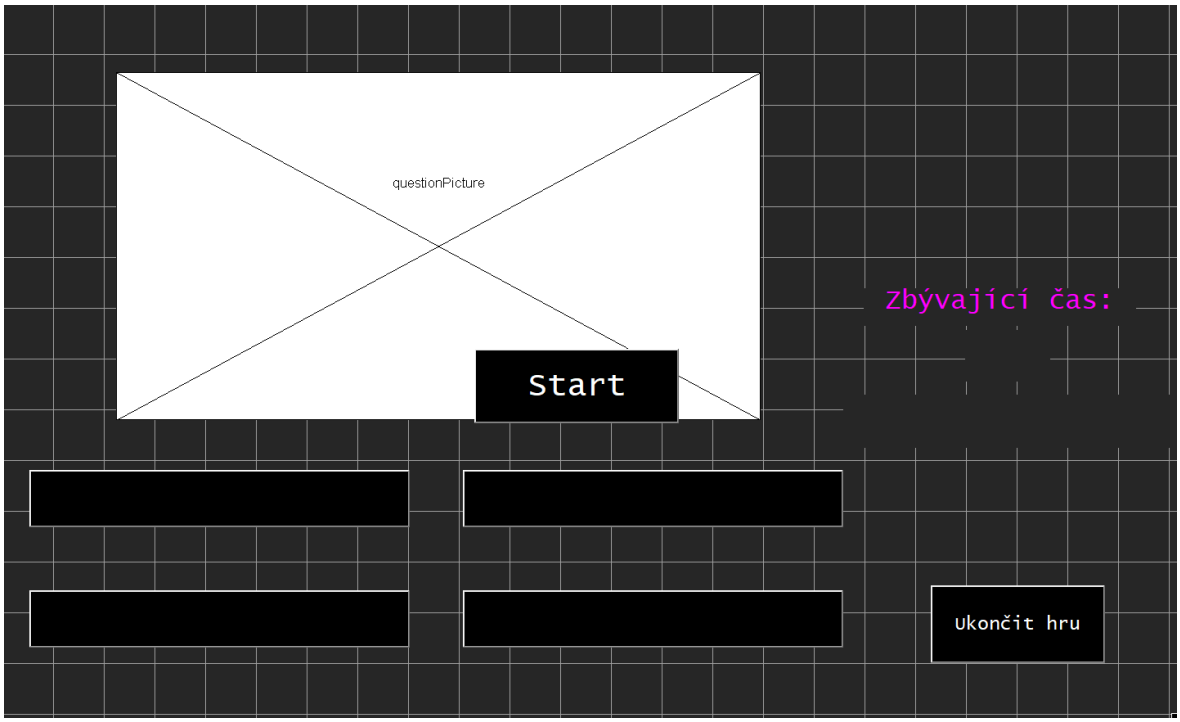
Tato kapitola se věnuje vývoji hry z pozice programátora.

5.1 Popis tvorby

Před začátkem praktické části jsem se seznámil s jinými bakalářskými pracemi na podobné téma, tyto práce jsou již popsány v teoretické části této práce. Z těchto prací jsem čerpal inspiraci.

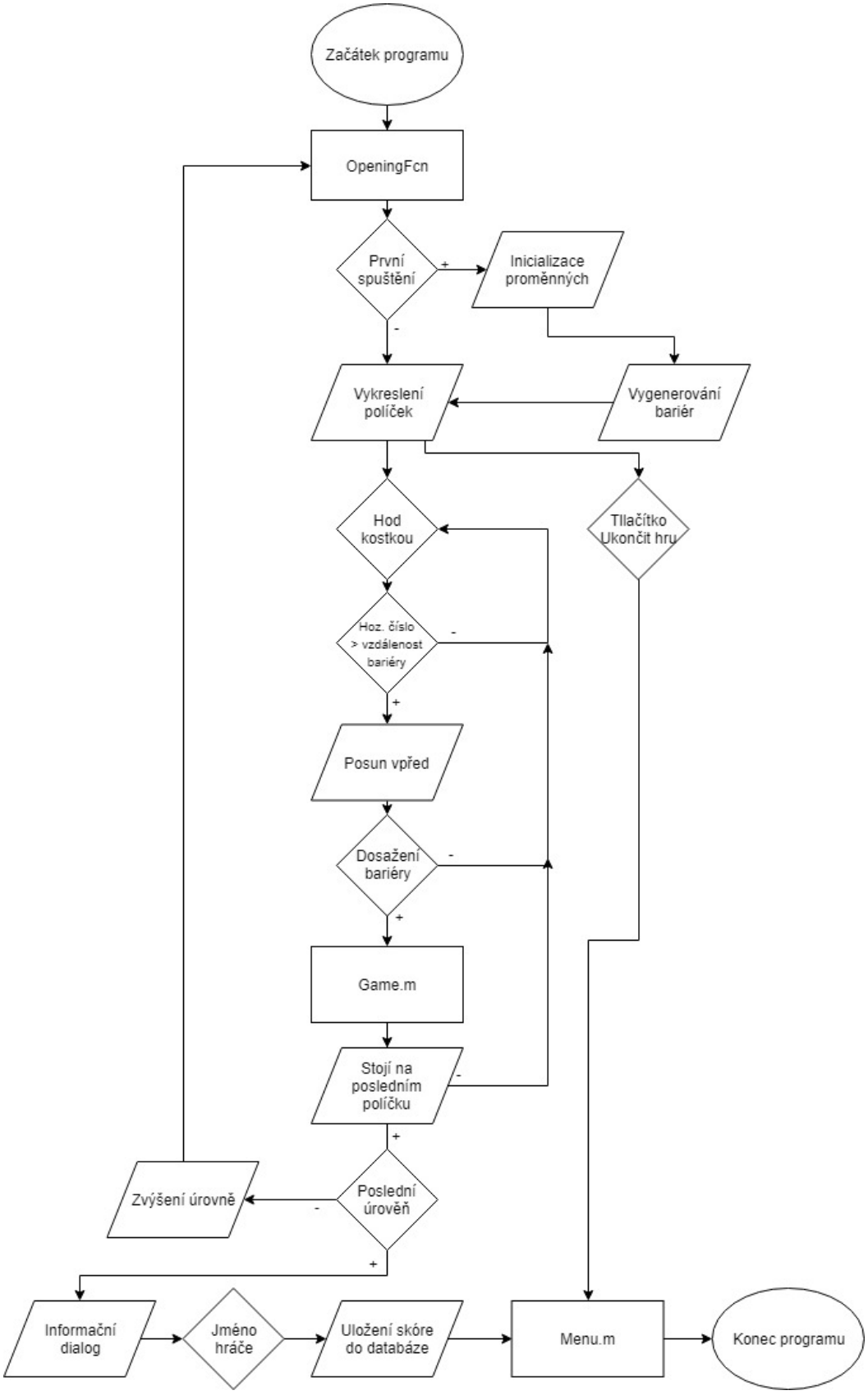
Dále jsem vytvořil návrh aplikace, který jsem při samotném vývoji neustále rozšiřoval na základě poznatků, které jsem při vývoji získal. Aplikaci jsem vyvíjel v programu MATLAB verze R2018a.

Nejdříve jsem vytvořil jedno MATLAB GUI, které se skládalo z jednoho FIG a jednoho M souboru. Toto GUI jsem v průběhu neustále přetvářel a optimalizoval až do stavu, kdy začalo ubývat kódu, který zde byl obsažený.

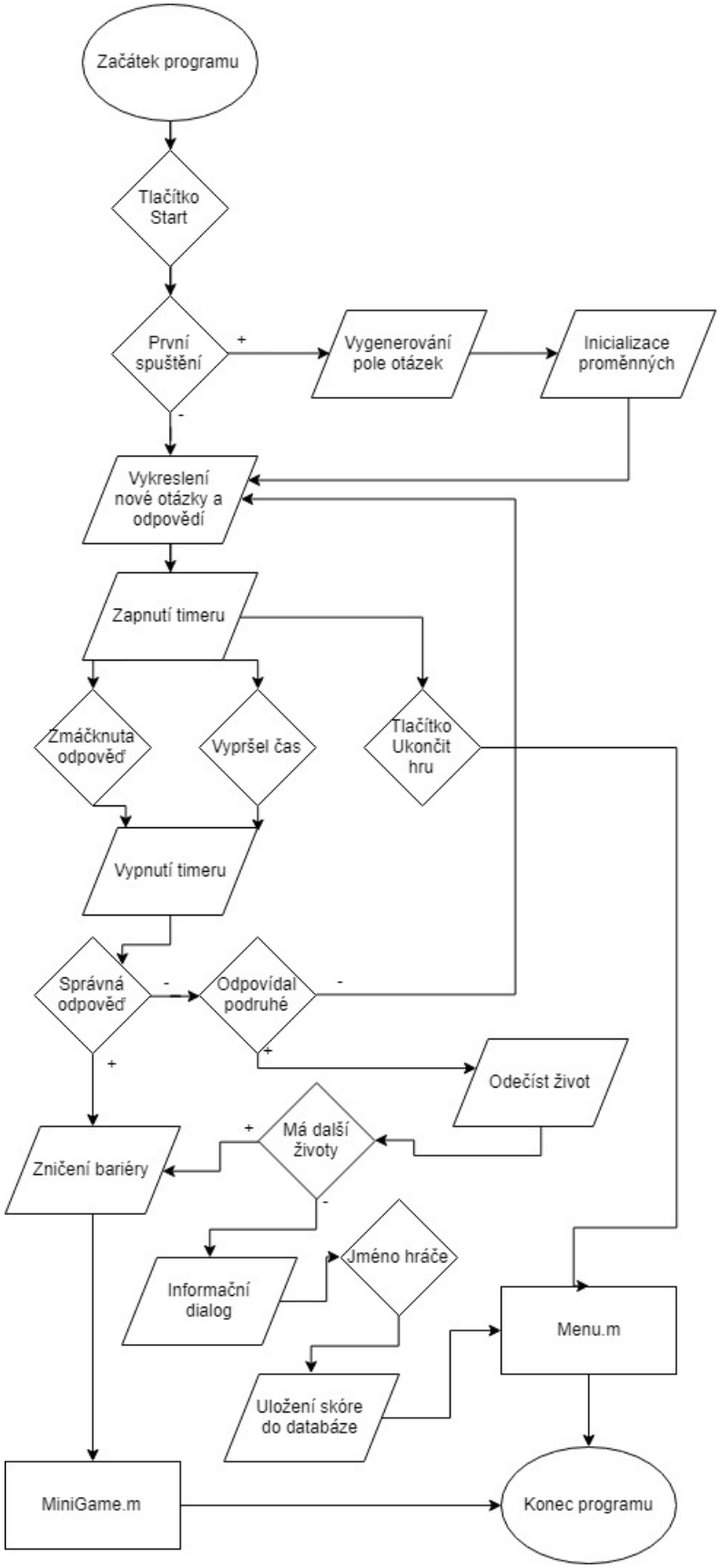


Obrázek 9 Náhled *Game.m* v GUIDE

Dva hlavní soubory GUI jsou pojmenované *MiniGame* (část s pohybem po políčcích) a *Game* (část s otázkami). Tyto soubory jsou vzájemně propojeny. Jejich struktura je popsána následujícími vývojovými diagramy.



Obrázek 10 Vývojový diagram souboru *MiniGame.m*



Obrázek 11 Vývojový diagram souboru *Game.m*

5.1.1 Databáze

Program využívá databázi ve formě SQL serveru. V této databázi jsou tři tabulky. Databáze je vyexportována v souboru *CreateDatabase.sql* a postup jejího vytvoření je popsán v souboru *VytvoreniDatabase.txt*.

5.1.1.1 Tabulka otázek a odpovědí

Tabulka otázek a odpovědí je stěžejní pro chod programu. Jsou zde uloženy otázky a odpovědi, které se využívají v hlavní části hry. Všechny otázky jsou z oblasti MATLABu a byly vytvořeny na základě mých znalostí programu a případného dohledávání na internetu.

Results Messages

	QuestionID	QuestionText	RightAnswer	WrongAnswer1	WrongAnswer2	WrongAnswer3	ImageUrl
1	1	Jakým príkazem v MATLABu vytvoríme tento graf?	plot(sin(-pi: 0.01: pi))	plot(tg(-pi: 0.01: pi))	plot(sinus(-pi: 0.0...	sin(-pi: 0.01: pi)	\images\question01.jpg
2	2	Jakým príkazem v MATLABu vytvoríme tento graf?	plot((-pi/2)+0.01:0.01...	plot((-pi/2)+0.01:0.0...	plot(tan(pi))	plot(sin(pi))	\images\question02.jpg
3	3	Jakým príkazem v MATLABu vytvoríme tento graf?	plot(cos(-pi: 0.01: pi))	plot(tan(-pi: 0.01: pi))	plot(cosinus(-pi: ...	plot(cos(pi))	\images\question03.jpg
4	4	Jakým príkazem lze porovnat stringy?	strcmp()	stringcomp()	strcomp()	cmpstr()	\images\question04.jpg
5	5	Jakým príkazem lze pozastavit program na 1 vterinu?	pause(1)	stop(1)	pause 1	stop 1	\images\question05.jpg
6	6	Jakým príkazem lze vytvorit pole plné nul?	zeros()	zero()	nulls()	[] = zeros	\images\question06.jpg
7	7	Jakým príkazem lze vytvorit pole plné jednicek?	ones()	one()	[1]	[] = ones	\images\question07.jpg
8	8	Jakým príkazem lze spojit stringy?	strcat()	strjoin()	join()	stringjoin()	\images\question08.jpg
9	9	Jakým znakem znacíme logický soucet v podmínce?	||	|	&&	&	\images\question09.jpg
10	10	Jakým znakem znacíme logický soucin v podmínce?	&&	&	||	|	\images\question10.jpg

Obrázek 12 Ukázka tabulky *Questions*

Tabulka je tvořena ze sedmi sloupců, z nichž *QuestionID* je primární klíč, který je jedinečný a automaticky generovaný. Dále jsou zde dva sloupce týkající se otázky – *QuestionText* a *ImageUrl*. Text otázky je uložen přímo v obrázku, ale pro lepší orientaci byl uložen i do samostatného sloupce. V dalších sloupcích jsou uloženy odpovědi na dané otázky.

Při prvním spuštění souboru Game.m v dané hře se otázky vytáhnou z databáze do programu pomocí příkazu:

```
curs = exec(conn,['SELECT [QuestionID], '...
'[QuestionText], [RightAnswer],[WrongAnswer1], ' ...
'[WrongAnswer2], [WrongAnswer3], [ImageUrl] ' ...
'FROM UTB_Bc.dbo.Questions']);
```

Následně se pomocí cyklu *for* naplní pole s objektů typu *Question*. Toto pole se zamíchá díky příkazu *randperm*. Před každým vykreslením nové otázky ve hře se navíc ještě zamíchají i odpovědi. Hráči se tedy nikdy nestane, že by narazil na stejné pořadí otázek se správnou odpovědí na stejném tlačítku.

5.1.1.2 Tabulky výsledků a hráčů

Další dvě tabulky jsou společně propojeny. Tabulka nejlepších výsledků s názvem *Scores* je pomocí cizího klíče *UserID* napojena na tabulku hráčů – *Users*. Toto propojení zajišťuje unikátnost každého jména (tato logika je řešena v programu).

Results Messages

	ScoreID	UserID	WrongAnswers	CorrectAnswers	Level	Points
1	1	6	2	0	1	40
2	2	6	3	1	1	55
3	3	6	2	0	1	40
4	4	6	2	0	1	40
5	5	6	2	0	1	40
6	6	6	2	0	1	40

Obrázek 13 Ukázka tabulky *Scores*

Results Messages

	UserID	Name
1	1	No Name
2	2	pepa
3	3	Honza
4	4	Pavel
5	5	Test
6	6	ondra

Obrázek 14 Ukázka tabulky *Users*

Logika pro získávání výsledků a hráčů z databáze je velmi podobná jako u otázek a odpovědí. Skóre a hráči však v databázi nejsou pevně daní, je třeba je tam vložit. Ke vložení je třeba nejdříve v MATLABu vytvořit tabulku se stejnými sloupci, jako jsou v tabulce v databázi a vložit do ní potřebná data.

```
data = table(score.UserID, score.WrongAnswers, ...
    score.CorrectAnswers, score.Level, score.Points, ...
    'VariableNames',{'UserID' 'WrongAnswers' ...
    'CorrectAnswers' 'Level' ,'Points'});
```

Tato data následně můžeme zapsat do databáze pomocí příkazu *sqlwrite*.

```
tableName = 'Scores';

sqlwrite(conn, tableName, data);
```

5.2 Funkce souborů

Aplikace je tvořena ze tří FIG souborů a sedmnácti M souborů. Každý FIG soubor je spojen s jedním M souborem stejného názvu, tyto soubory tvoří jádro celé aplikace, ostatní soubory plní pouze podpůrné funkce a jsou rozděleny podle svého využití. Mezi největší skupinu souborů patří soubory pro práci s databází nacházející se v adresáři *SQLScripts*. Data, která se posílají nebo přijímají z databáze, jsou ukládána do objektů vytvořených pomocí *class* (tříd), tyto soubory jsou uloženy v adresáři *Models*.

5.2.1 **Question.m, Score.m, User.m**

Všechny tři soubory slouží jako třídy pro vytváření objektů. Každý soubor reprezentuje stejnojmennou tabulku v databázi a obsahuje několik *properties*, které kopírují sloupce v daných tabulkách.

5.2.2 **OpenDBConnection.m**

Jedná se o soubor s jedinou funkcí stejného názvu. Byl vytvořen za účelem minimalizace opakujícího se kódu. Kód, který je zde obsažený, je potřebný při každém využití databáze, opakoval by se tedy v několika jiných souborech. Samotným úkolem tohoto souboru je nastavení a vytvoření připojení k databázi.

5.2.3 **CloseDBConnection.m**

Vznikl ze stejného důvodu jako OpenDBConnection.m. Úkolem tohoto souboru je uzavřít připojení k databázi a vyčistit vytvořené proměnné.

5.2.4 **GetQuestions.m**

Název funkce	Pozice v souboru	Účel
GetQuestions	1 – 25	Naformátování dat pro následnou práci v programu, zamíchání otázek
getQuestionsFromDatabase	27 – 35	Získání dat z databáze

Tabulka 2 Funkce souboru *GetQuestions.m*

Hlavní funkce *GetQuestions* volá funkce pro vytvoření připojení, získání dat a uzavření připojení databáze. Následně získanými daty naplní pole objektů třídy *Question* a pole zamíchá pomocí funkce:

```
shuffledArray =
    questionArray(randperm(length(questionArray)));
```

5.2.5 **GetScores.m**

Název funkce	Pozice v souboru	Účel
GetScores	1 – 27	Naplnění pole výsledků
getScoresFromDatabase	29 – 40	Získání dat z databáze

Tabulka 3 Funkce souboru *GetScores.m*

Funguje obdobně jako soubor *GetQuestions.m*. Hlavní funkce *GetScores* volá funkci *getScoresFromDatabase* k získání deseti nejlepších výsledků z databáze a následně s těmito výsledky naplní pole objektů třídy *Score*.

5.2.6 **SaveScore.m**

Název funkce	Pozice v souboru	Účel
SaveScore	1 – 10	Vytvoření tabulky pro uložení
saveScoreToDatabase	12 – 16	Uložení skóre

Tabulka 4 Funkce souboru *SaveScore.m*

Obě funkce jsou opět propojeny. Tentokrát se však provádí opačný proces než při získávání dat. Data se tedy musí nejdříve naformátovat do tabulky, která se následně pošle do databáze.

5.2.7 **GetUsers.m**

Název funkce	**Pozice v souboru**	**Účel**
GetUsers	1 – 20	Porovnání dat v databázi s daty přicházejícími z programu
getUsersFromDatabase	22 – 29	Získání dat z databáze

Tabulka 5 Funkce souboru *GetUsers.m*

Na první pohled se jedná o stejnou funkci jako u ostatních souborů začínajících s *Get*. Tentokrát je však po získání dat z databáze potřeba ověřit zda se tam už nachází jméno hráče, který právě dohrál. Pokud tomu tak je, vrátí se jeho informace zpět do programu. Pokud v databázi ještě není, tak se zavolá funkce *SaveUser*.

5.2.8 **SaveUser.m**

Název funkce	**Pozice v souboru**	**Účel**
SaveUser	1 – 12	Vytvoření tabulky pro uložení a nastavení objektu *User*
saveUserToDatabase	14 – 25	Uložení hráče a vrácení vygenerovaného ID z databáze

Tabulka 6 Funkce souboru *SaveUser.m*

Funkce *SaveUser* v první části vytvoří tabulku, do které se vloží záznam se jménem hráče, následně zavolá funkci *saveUserToDatabase*, která vykoná samotné uložení. Následně však ještě vrátí ID, jež se vygenerovalo při uložení a vrátí ho zpět do funkce *SaveUser*. V pokračování této funkce se vytvoří objekt třídy User, do kterého se přiřadí ID a jméno z databáze.

5.2.9 **Menu.m**

Jak už bylo řečeno, *Menu.m* je výchozí soubor celé aplikace, z jiného souboru se aplikaci nepodaří správně spustit, protože všechny soubory mezi sebou mají závislosti a vyžadují parametry, bez kterých je nelze spustit.

Název funkce	Pozice v souboru	Účel
Menu_OpeningFcn	23 – 38	Nastavení názvu, vyčištění proměnných z předchozí hry
rulesButton_Callback	47 – 51	Skrytí prvků menu a zobrazení prvků pravidel
leaderboardButton_Callback	54 – 78	Skrytí prvků menu, získání dat z databáze a zobrazení tabulky nejlepších hráčů
playButton_Callback	81 – 84	Přesměrování do souboru *Minigame.m*
returnButton_Callback	87 – 89	Zobrazení prvků menu a skrytí jiných prvků (pravidla, nejlepší hráči)
hideMenu	97 – 103	Skrytí prvků menu a zobrazení tlačítka *Zpět*
showMenu	103 – 116	Zobrazení menu a skrytí všech ostatních prvků

Tabulka 7 Funkce souboru *Menu.m*

Funkce *Menu_OpeningFcn* se spouští při každém spuštění souboru, jejím úkolem je uskutečnit příkazy, které je třeba provést předtím, než uživatel bude moci něco dělat. V tomto případě se jedná o vycentrování okna doprostřed obrazovky, nastavení nadpisu a především vymazání všech globálních proměnných. Vymazáním globálních proměnných se zabrání případným pádům aplikace kvůli špatným hodnotám.

Další funkcí je *callback* tlačítka *rulesButton*, tedy tlačítka pro zobrazení pravidel hry. Tato funkce pouze skrývá všechny prvky menu pomocí funkce *hideMenu*, zobrazí text pravidel a tlačítko *Zpět*.

Na stejném principu funguje i funkce *leaderboardButton_Callback*, tentokrát je však kromě pro naplnění tabulky nejlepších výsledků zavolána navíc funkce ze souboru *GetScores.m* pro získání deseti nejlepších výsledků z databáze. Tyto data jsou následně zpracovány do podoby, která se dá zobrazit pomocí tabulky.

Funkce *playButton_Callback* je pouhým přesměrováním na soubor *MiniGame.m*.

Funkce *hideMenu* a *showMenu* plní přesně opačné funkce a existují kvůli minimalizaci opakujícího se kódu. Minimálně jedna z nich se využívá při stisknutí téměř všech tlačítek.

ReturnButton_Callback plní opačnou funkci než funkce pro zobrazení pravidel a nejlepších výsledků, tedy skrývá prvky, které tyto funkce zviditelnily, a zobrazuje prvky, které byly skryty.

5.2.10 **MiniGame.m**

Název funkce	**Pozice v souboru**	**Účel**
MiniGame_OpeningFcn	23 – 66	Nastavení proměnných a prvků GUI
throwButton_Callback	81 – 154	Vygenerování hozeného čísla a porovnávání s pozicí další bariéry
drawCircles	157– 189	Vykreslení políček
prepareBarriers	192 – 205	Vygenerování indexů bariér
advanceToNextPosition	208 – 221	Pohyb po políčcích
setPositionAndPrepare-Barriers	232 – 236	Úvodní nastavení pozice a volání funkce *prepareBarriers*
setLivesCount	239 – 236	Zobrazení počtu životů
setThrownNumberImage	239 – 273	Vykreslení kostky

Tabulka 8 Funkce souboru *MiniGame.m*

Funkce MiniGame_OpeningFcn se stará o nastavení viditelnosti a pozice prvků *axes*, do kterých se vykreslují políčka a kostka. Dále se zde inicializují počáteční hodnoty úrovně, životů a také zde volá funkce *setPositionAndPrepareBarriers*. Další důležitou funkcí je ověření, zda hráč stoupnul na poslední políčko, za těchto okolností se volá funkce *SetScore* (v případě, že je ve čtvrté úrovni) nebo se zvyšuje úroveň. Navíc se zde volají funkce pro *drawCircles* a *setLivesCount*.

ThrowButton_Callback se volá v momentě, kdy uživatel zmáčkne tlačítko *Hodit kostkou*. Jako první se vygeneruje náhodné celé číslo od jedné do šesti pomocí funkce *randi*. Dále se zavolá funkce *setThrownNumberImage* pro vykreslení kostky na základě vygenerovaného čísla.

Pro vykreslení políček se využívá funkce drawCircles. V této funkci je vytvořený cyklus for k vykreslení 25 políček, jejichž pozice jsou uloženy v poli. V každém průchodu cyklu se vykresluje jedno políčko, u kterého se kromě pozice mění ještě barva, a to na základě proměnné *currentPosition*(určuje aktuální pozici hráče) a pole *barrierIndexes* (vygenerované pozice bariér).

```
if ismember(i, barrierIndexes)
     rectangle('Position', posArray(i, :), 'Curvature', ...
     [1 1], 'FaceColor', 'b', 'EdgeColor', 'w');
elseif currentPosition == i
     rectangle('Position', posArray(i, :), 'Curvature', ...
     [1 1], 'FaceColor', 'y', 'EdgeColor', 'w');
else
     rectangle('Position', posArray(i, :), 'Curvature', ...
     [1 1], 'FaceColor', 'k', 'EdgeColor', 'w');
end
```

Bariéry se generují ve funkci *prepareBarriers*, která využívá funkce *randperm* k vygenerování pole bariér, jejichž množství se mění na základě úrovně.

Funkce *advanceToNextPosition* je jednoduchým způsobem jak zanimovat pohyb po cestě. Hlavní částí funkce je cyklus *while*, který v každém průchodu zvyšuje aktuální pozici hráče a používá funkci *drawCircles* k opětovnému vykreslování políček s novou pozicí hráče. Tento cyklus končí v momentě, kdy hráč dosáhne pozice, kdy se požadovaná pozice rovná aktuální pozici hráče.

```
nextPosition = currentPosition + thrownValue;

while nextPosition ~= currentPosition

     currentPosition = currentPosition + 1;

     pause(0.5)

     drawCircles(handles);

end
```

Následují dvě jednoduché funkce. První z nich je *setPositionAndPrepareBarriers*, ta se volá při spuštění programu z funkce *MiniGame_OpeningFcn* a má za úkol nastavit pozici hráče na 1 a zavolat funkci *prepareBarriers*.

Poslední funkcí v souboru *MiniGame.m* je *setThrownNumberImage*, která funguje na stejném principu jako *drawCircles*, nevykresluje však políčka, ale kostku s hozeným číslem. Rozhodování o počtu teček na kostce je řešeno pomocí *switche* se šesti variantami. Každá z těchto variant vytváří pole s jinými pozicemi teček na kostce.

5.2.11 **Game.m**

Název funkce	**Pozice v souboru**	**Účel**
playButton_Callback	44 – 69	Nastavení proměnných a prvků GUI
answerButton_Callback	72 – 89	*Callbacky* čtyř tlačítek s odpovědí, volá se zde *questionAnswered* s parametrem závislým na čísle tlačítka
isCorrectAnswer	103 – 157	Porovnání indexu správné odpovědi s indexem zmáčknutého tlačítka a nastavení pozadí prvků na základě výsledku porovnání

setAnswerFields	160 – 165	Nastavení textů odpovědí
play	168 – 217	Zobrazení otázky, zamíchání odpovědí, spuštění timeru
questionAnswered	220 – 232	Vypnutí tlačítek s odpověďmi, vypnutí timeru, volání *isCorrectAnswer*
timerCallback	235 – 251	Vykreslení zbývajícího času, detekce vypršení času
startTimer	254 – 262	Nastavení a spuštění timeru
isSecondQuestion	265 – 282	Ověření zda uživatel chyboval dvakrát po sobě

Tabulka 9 Funkce souboru *Game.m*

Úvodní funkcí souboru *Game.m* je *callback* tlačítka *Hrát – playButton_Callback*. Se zmáčknutím tohoto tlačítka se spustí nekonečné volání funkcí, které končí až v momentě, kdy je soubor zavřen. Nejdříve se vykoná podmínka, která inicializuje proměnné v případě, že je prázdné pole otázek. To značí, začátek hry, inicializují se tedy některé proměnné a volá se funkce *GetQuestions*, která vrátí zamíchané pole otázek. Dále se zviditelní všechny prvky tohoto GUI a zavolá se hlavní funkce *play*.

V souboru jsou čtyři *callbacky* pro čtyři různé *answerButton*, které se liší pouze číslem. Každý z nich plní stejnou funkci – volání *questionAnswered*, vždy však s číslem tlačítka, které bylo právě zmáčknuto.

Funkce *isCorrectAnswer* je nejobsáhlejší funkcí souboru. Na základě indexu správné a špatné odpovědi se zde vykonává kód pro podbarvení těchto tlačítek zelenou, respektive červenou barvou. Hlavní částí je však porovnání indexu správné odpovědi a indexu tlačítka, který sem doputoval až z *answerButton_Callback*. Pokud je odpověď správně, tak se inkrementuje množství správných odpovědí, smaže se bariéra, na které hráč stál a na danou pozici se dosadí jeho aktuální pozice. V případě, že je odpověď špatně, tak se ještě navíc volá funkce *isSecondQuestion* pro zjištění zda se má zobrazit další otázka nebo odečíst život.

Jako hlavní funkce programu by se dala označit funkce *play*, v této funkci se inicializuje *timer*, nastavuje se zde zbývající čas a především se zde náhodně nastavují pozice odpovědí. Zamíchání odpovědí je v první fázi řešeno vygenerováním náhodného indexu správné odpovědi. Následně se zbývající (špatné) odpovědi vloží do pole a zamíchají se pomocí funkce *randperm*. Dále se využije propojení s funkcí *setAnswerFields*, jíž úkolem je jednoduché nastavení textů u tlačítek s odpověďmi. Jednoduchost této funkce je umožněna pomocí *switche* ve funkci *play*, tento *switch* posílá do funkce *setAnswerFields* čtyři parametry na základě vygenerovaného indexu správné odpovědi. Na konci funkce se zavolá funkce *startTimer*.

StartTimer je funkce jejíž název mluví sám za sebe. Jejím úkolem je zapnutí *timeru*.

```
t.StartDelay = 0;

t.TasksToExecute = timeLeft;

t.ExecutionMode = 'fixedRate';

t.TimerFcn = {@timerCallback, handles};

start(t);
```

V tomto kódu se nastavují parametry již vytvořeného objektu s názvem *t*, což je objekt *timer*. Parametru *TasksToExecute* se přiřazuje proměnná *timeLeft*, ve které je uložen čas, po který má *timer* běžet. V parametru *TimerFcn* se volá funkce *timerCallback*, což je manuálně vytvořený *callback*, tedy funkce, která se bude vykonávat, dokud bude *TasksToExecute* větší než 0.

TimerCallback již byl zmíněný v předchozím textu. V této funkci je kód, který se vykonává každou vteřinu až do vypnutí *timeru*. Proměnná *timeLeft* se při každém spuštění funkce dekrementuje o 1. V momentě, kdy je *timeLeft* roven nule, dochází k jeho vypnutí (příkazem *stop*), smazání této instance (příkazem *delete*) a volání funkce *isSecondQuestion*.

Funkce *isSecondQuestion* je tvořena jednou velkou podmínkou. V případě, že již hráč odpovídal podruhé za sebou, je odečten život, smazána bariéra a nastavena nová aktuální pozice *hřáče* a vypnuto GUI Game, hráč se vrací zpět do *MiniGame*. V opačném případě se volá funkce play a hráč dostává možnost odpovídat na jinou otázku.

5.2.12 **Login.m**

Tento soubor se volá, když končí hra. Jeho úkolem je obeznámit hráče o konci hry a získat jeho jméno. V případě, že hráč jméno nezadá nebo zadá prázdný *string*, uloží se do databáze jako *NoName.*

5.2.13 **SetScore.m**

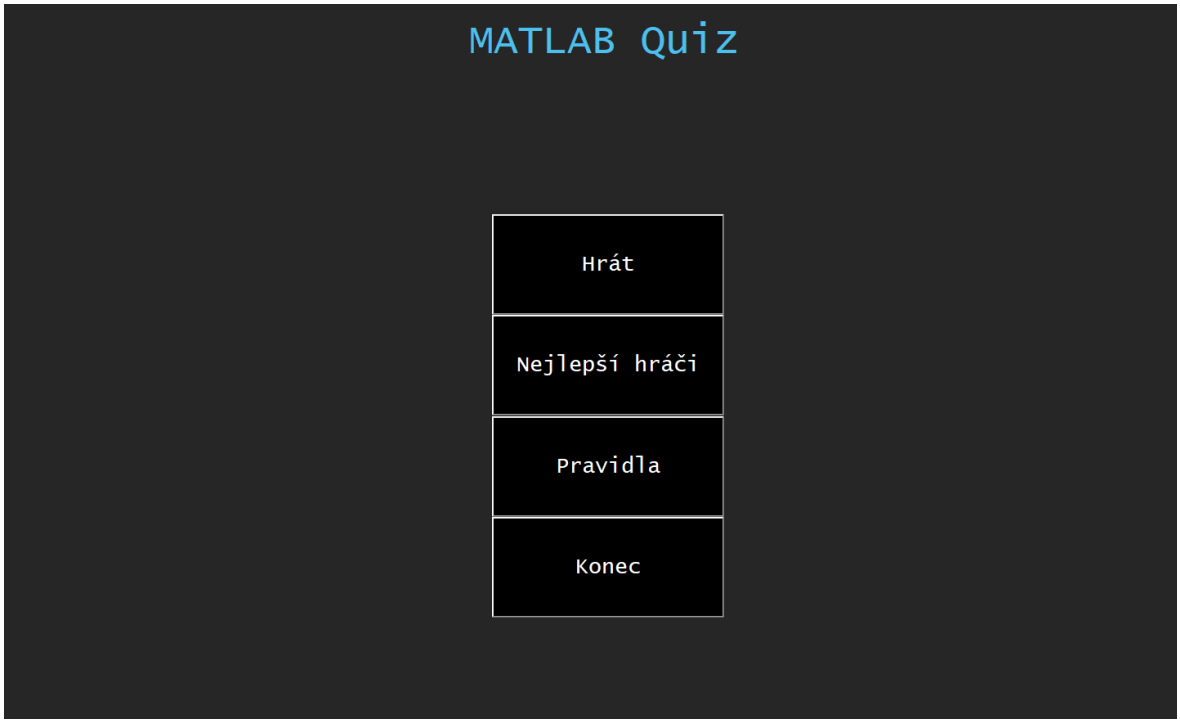
Další soubor využívaný při konci hry. Na začátku volá *Login.m* a následně se vytvoří objekt *Score*, do kterého se přiřadí hodnoty získané ze hry (správné a špatné odpovědi, dosažená úroveň).

6 APLIKACE Z POHLEDU UŽIVATELE

Aplikaci může uživatel spustit pomocí příkazu *Menu*.

6.1 Menu

Po spuštění aplikace je uživateli zobrazeno úvodní okno v podobě menu se čtyřmi položkami.



Obrázek 15 Ukázka Menu aplikace

Každá položka je zpracována reprezentována tlačítkem, které uživatel může stisknout najetím kurzoru a zmáčknutím levého tlačítka myši. Tlačítko *Konec* program ukončí. Ostatní tlačítka budou popsána v následujících kapitolách.

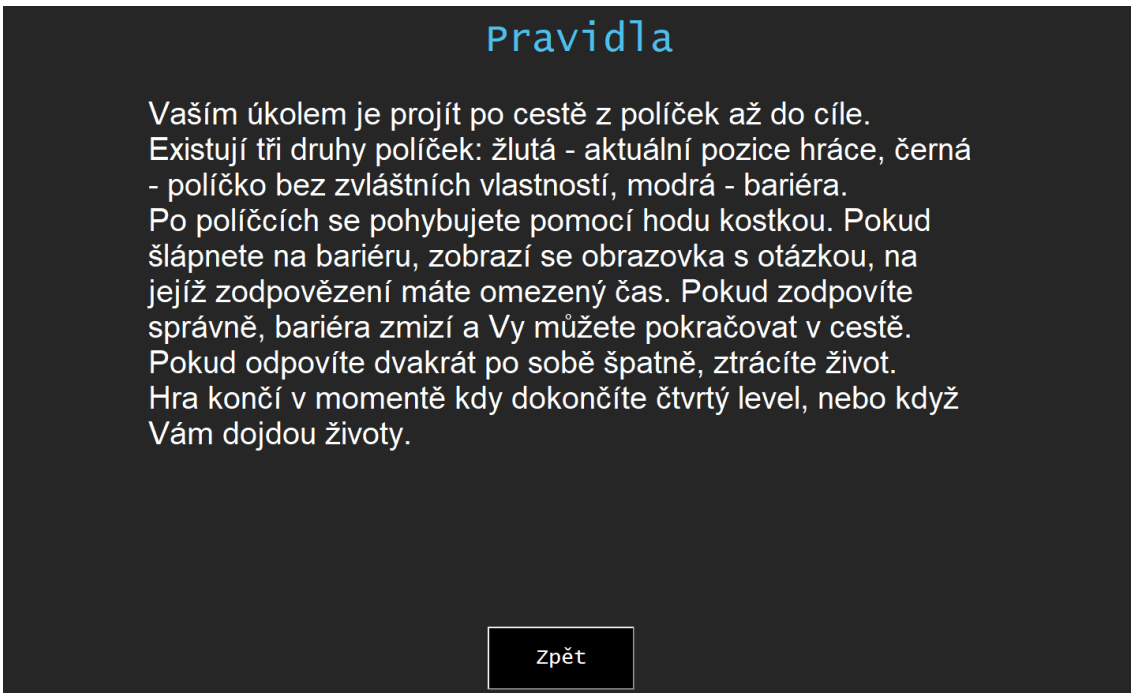
6.2 Nejlepší hráči

Stisknutím tlačítka *Nejlepší hráči* je uživatel přesměrován na obrazovku s tabulkou nejlepších hráčů. Tato tabulka obsahuje čtyři sloupce a je zde zobrazeno až 10 záznamů seřazených podle nejvyššího počtu bodů. Jediným tlačítkem na této obrazovce je tlačítko *Zpět*, které uživatele vrátí zpět do *Menu*.

Obrázek 16 Ukázka nejlepších výsledků

6.3 Pravidla

Zde si uživatel může přečíst pravidla hry. Na této obrazovce má uživatel opět jedinou možnost, na kterou může kliknout – tlačítko *Zpět*, to jej opět vrátí zpět do hlavního *Menu*.

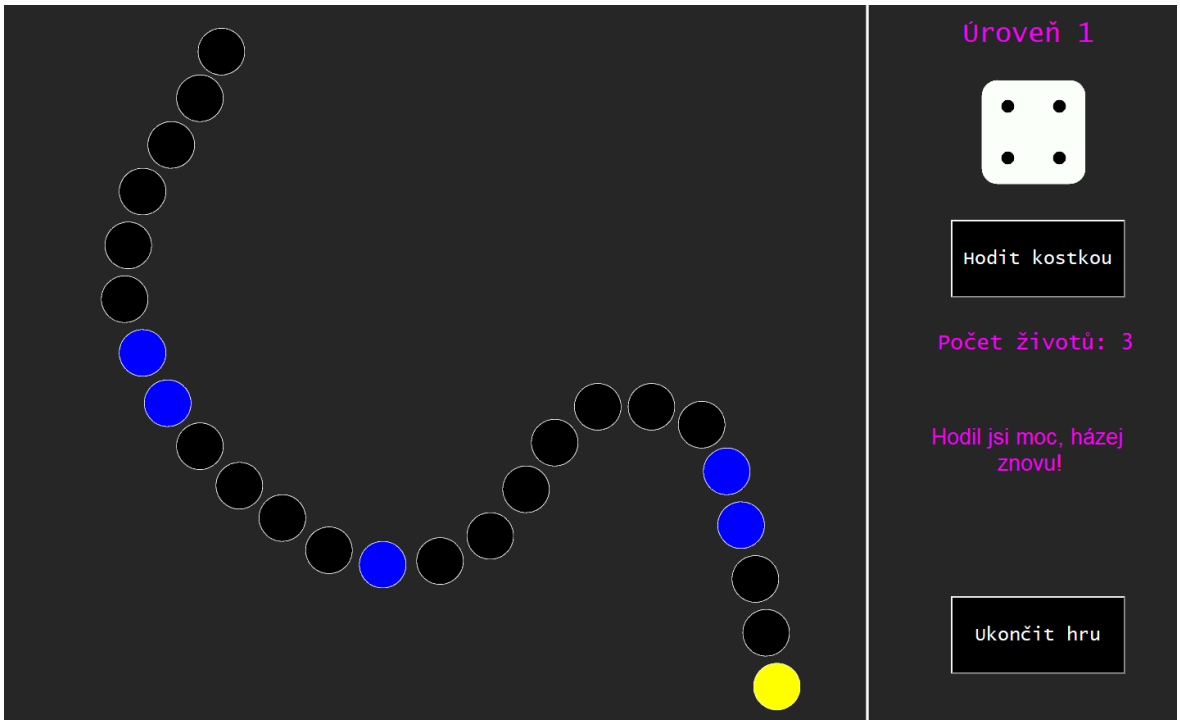


Obrázek 17 Ukázka pravidel ve hře

6.4 Hrát

Po stisku tlačítka Hrát se vypne aktuální GUI a zapne se nové. Na nové obrazovce může hráč vidět po levé straně dvacet pět políček s různými barvami. Vlastnosti těchto políček jsou vysvětleny v pravidlech hry. U pravého okraje okna hráč vidí dvě tlačítka a dva informační řádky. Dozví se zde na jaké je úrovni a kolik mu zbývá životů. Pomocí tlačítka Ukončit hru se může opět vrátit zpět do Menu.

Tlačítko Hodit kostkou může způsobit několik akcí. Při každém stisku tohoto tlačítka se vykreslí kostka s počtem teček, který reprezentuje hozené číslo. Pokud je číslo větší, než je vzdálenost nejbližší bariéry, zobrazí se nové textové pole, které uživatele vyzve, aby házel znovu. Pokud hodí méně než je vzdálenost k nejbližší bariéře, začne se postupně pohybovat po políčcích směrem kupředu. V momentě, kdy šlápne na bariéru, aktuální okno se vypne a otevře se nové.



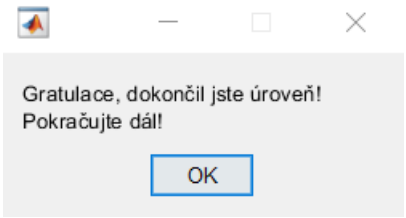
Obrázek 18 Ukázka okna s políčky po hodu kostkou

V novém okně uživatel vidí obrázek s otázkou (která se může i nemusí týkat daného obrázku) a čtyři tlačítka s odpovědí. Stejně jako v předchozím okně se i zde objevuje tlačítko Ukončit hru. Novinkou je textové pole Zbývající čas, toto pole uživateli říká, kolik času mu zbývá na zodpovězení otázky.

Pokud nestihne v časovém limitu otázku zodpovědět, je otázka vyhodnocena jako špatně zodpovězená. Jestliže odpověděl špatně nebo vypršel čas, dostane náhradní otázku, časový limit se opět vrátí zpět na původní hodnotu. Pokud se mu znovu nepovede odpovědět správně, ztratí jeden život, okno se zavře a otevře se opět předchozí okno s políčky, uživatel teď zaujal pozici, na které předtím byla bariéra. V případě, že se mu otázku povede zodpovědět správně a v časovém limitu, také se vrací zpět na předchozí obrazovku, tentokrát mu však zůstává stejný počet životů.

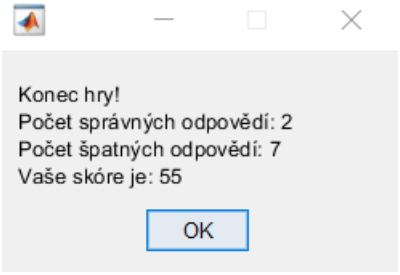
Obrázek 19 Ukázka okna s otázkou

Dosažením posledního políčka uživatel vyvolá informační dialog, po odkliknutí tohoto dialogu je uživatel přesunut do další úrovně. Tentokrát na něj čeká více bariér a méně času na zodpovězení otázek.



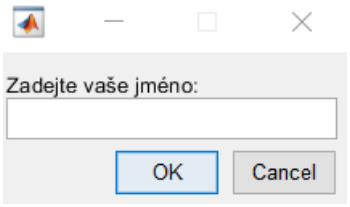
Obrázek 20 Dialog, informující o dokončení úrovně

Jestliže se uživatel dostane až do čtvrté úrovně a podaří se mu projít přes všechny bariéry a stoupnout si na poslední políčko, zobrazí se dialog, informující o počtu správných a špatných odpovědí a také dosaženém počtu bodů.



Obrázek 21 Dialog s informacemi o konci hry

Po kliknutí na tlačítko *OK* vyskočí další dialog. Tentokrát je uživatel vyzván k vyplnění svého jména. Jestliže uživatel jméno nevyplní a klikne na tlačítko *OK* nebo *Cancel*, program to rozpozná a uloží za jeho jméno *No Name*.



Obrázek 22 Dialog dotazující se na jméno hráče

Tyto informace se následně uloží do databáze a jsou zobrazeny v tabulce nejlepších výsledků, je-li počet bodů mezi deseti nejlepšími.

Po zavření dialogu pro vyplnění jména je opět otevřeno *Menu*.

ZÁVĚR

Cílem práce bylo vytvoření interaktivní aplikace s otázkami z oblasti MATLAB. Aplikace byla zpracována do zábavné výukové formy a dostala název MATLAB Quiz. V aplikaci se uživatel pohybuje po cestě vymezené políčky, kde při sešlápnutí speciálního políčka – tzv. bariéry dostane otázku z oblasti MATLABu. Po zodpovězení této otázky může pokračovat dále.

V teoretické části práce jsou popsány funkce prostředí MATLAB a jeho programovacího jazyka. Po přečtení této části by uživatel měl mít o tomto prostředí a programovacím jazyku dostatečný přehled na to, aby sám dokázal vytvořit jednoduché aplikace. Dále byly v teoretické části popsány jiné interaktivní aplikace, které byly zpracovány v podobě bakalářských prací v předchozích letech.

V praktické části je popsán vývoj samotné aplikace, její vývojové diagramy, propojení s SQL databází a funkce souborů, na nichž aplikace stojí. Druhá kapitola praktické části se věnuje popisu výsledné aplikace s uživatelského hlediska.

Hlavní část aplikace je tvořena databází čítající 100 otázek z oblasti MATLABu a je určena pro uživatele s dobrými znalostmi tohoto prostředí a jeho programovacího jazyka. Aplikace může sloužit jako výukový materiál pro studenty, kteří si s její

SEZNAM POUŽITÉ LITERATURY

Monografie:

[1] KARBAN, Pavel. Výpočty a simulace v programech Matlab a Simulink. Brno: Computer Press, 2006. ISBN 978-80-251-1448-3.

[2] ZAPLATÍLEK, Karel a Bohuslav DOŇAR. MATLAB: tvorba uživatelských aplikací. Praha: BEN - technická literatura, 2004. ISBN 80-7300-133-0.

[3] DUŠEK, František. MATLAB a SIMULINK: úvod do používání. Pardubice: Univerzita Pardubice, 2000. ISBN 80-7194-273-1.

[4] HECZKO Michal. Výukový a zkušební program pro předmět PPAŘ. Bakalářská práce. Zlín: Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky, 2006

[5] KOZÁK, Štefan a Slavomír KAJAN. Matlab - Simulink. 1. vyd. Bratislava: Slovenská technická univerzita, 1999. ISBN: 80-227-1213-2

Internetové zdroje:

[6]MATLAB. *Wikipedia* [online]. [cit. 2019-04-25]. Dostupné z: https://en.wikipedia.org/wiki/MATLAB

[7] MATLAB Documentation. *MathWorks* [online]. [cit. 2019-04-25]. Dostupné z: https://www.mathworks.com/help/matlab/

[8] Introduction to GUI Callbacks. *MATLAB Webserver* [online]. [cit. 2019-04-25]. Dostupné z: http://matlab.izmiran.ru/help/techdoc/creating_guis/callbacks2.html

[9] SQL Standards. *Oracle Help Center* [online]. [cit. 2019-04-25]. Dostupné z: https://docs.oracle.com/cd/B13789_01/server.101/b10759/intro002.htm

[10] Interactive Online SQL Training. *SQLCourse* [online]. [cit. 2019-04-25]. Dostupné z: http://www.sqlcourse.com/

[11] Configure, explore, and import database data. *MathWorks* [online]. [cit. 2019-04-25]. Dostupné z: https://www.mathworks.com/help/database/ug/databaseexplorer-app.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MATLAB	Matrix Laboratory
GUI	Graphical User Interface
SQL	Structured Query Language
GUIDE	Graphical User Interface Development Environment

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD-ROM

fulltext.pdf – text bakalářské práce ve formátu PDF

prilohy.zip – soubory obsahující zdrojový kód aplikace